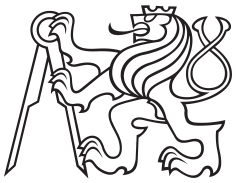
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Logovací systém pro nástroj na výuku transformací I3T

Filip Uhlík

Školitel: Ing. Petr Felkel, Ph.D.
Studijní program: Software inženýrství a technologie
Květen 2020

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Petru Felkelovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 22. května 2020

Abstrakt

Transformace v počítačové grafice dělají studentům potíže. S pochopením transformací pomáhá Interaktivní nástroj na výuku transformací (I3T), který je využíván na Katedře počítačové grafiky a interakce v předmětu Programování grafiky. Cílem této práce je vytvořit pro I3T nástroje, které pomohou s dalším vývojem aplikace jako takové a jejího obsahu (naučné vzorové úlohy), popřípadě pomohou zjistit, na co se zaměřit během výuky. Tyto nástroje jsou logovací systém uživatelské interakce v aplikaci I3T a aplikace, která záznamy pomůže vyhodnotit.

Do nástroje I3T byl implementován logger, který pomocí sledování událostí (tzv. Event tracking) umožňuje zaznamenávat uživatelskou interakci do souboru. Pro vyhodnocení záznamů byla navržena a poté v programovacím jazyce Python implementována vyhodnocovací aplikace Log Viewer. Aplikace zobrazuje záznamy interakce jako časové osy.

Logovací systém společně s aplikací Log Viewer umožňuje snadné získání a vyhodnocení záznamů uživatelské interakce v aplikaci I3T. Log Viewer umožňuje také porovnávat záznamy mezi sebou nebo se vzorovým řešením úlohy. Logování bude sloužit k vývoji aplikace I3T, vývoji vzorových úloh, které jsou součástí I3T, a k zlepšení zaměření výuky studentů.

Klíčová slova: logování, uživatelská interakce, i3t, python, vizualizace

Školitel: Ing. Petr Felkel, Ph.D.

Abstract

Transformations in computer graphics are difficult for the students. The Interactive Tool for Teaching Transformations (I3T), which is used at the Department of Computer Graphics and Interaction in the subject of Graphic Programming, helps to understand these transformations. The aim of this work is to create tools for I3T that will help with the further development of the application and its content (educational tutorial tasks), or help to find out what to focus on during teaching. These tools are the user interaction logging system in the I3T application and the application that will help evaluate the records.

A logger has been implemented in the I3T tool, which allows event tracking to record user interaction to a file. An evaluation application called Log Viewer was designed and then implemented in the Python programming language. The application displays interaction records as timelines.

The logging system together with the Log Viewer application allows easy acquisition and evaluation of user interaction records in the I3T tool. Log Viewer also allows you to compare logs with each other or with an ideal solution. Logging will be used to develop the I3T tool, to develop the tutorial tasks that are part of I3T, and to improve teaching focus.

Keywords: logging, user interaction, i3t, python, visualisation

Title translation: Logging system for the Interactive Tool for Teaching Transformations - I3T

Obsah

Obrázky

Tabulky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Uhlík** Jméno: **Filip** Osobní číslo: **474489**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Logovací systém pro nástroj na výuku transformací I3T

Název bakalářské práce anglicky:

Logging system for the Interactive Tool for Teaching Transformations - I3T

Pokyny pro vypracování:

Cílem práce je pro nástroj na výuku transformací I3T, který je vyvíjen na katedře 13139, vytvořit hierarchický logovací sytém pro zaznamenávání interakce uživatele. Systém bude využíván při výzkumu postupů výuky transformací a při uživatelském testování nástroje I3T.
Analyzujte dostupné metody a nástroje pro testování uživatelské interakce desktopových aplikací. Sestavte požadavky na logovací systém v případě aplikace I3T (a) pro vývoj aplikace, (b) pro sledování řešení úloh (pro úlohy v I3T) a vybere nejvhodnější postup implementace.
Strana I3T:
- stanovte typické testované situace a pro ně požadovaný způsob logování
- zvolte vhodnou knihovnu pro implementaci
- logovací funkce integrujte v I3T
vyhodnocovací aplikace
- vytvořte aplikaci pro prohlížení a vyhodnocování naměřených dat, aplikace bude využívat hierarchii akcí (úloha – logické kroky – interakce s I3T).
Navrhněte postup vytváření testovacích scénářů v případech (a) i (b) a ukažte různé způsoby vyhodnocení interakce. Výslednou dvojici – logovací knihovna – vyhodnocovací aplikace otestujte na třech úkolech (dvou existujících tutoriálech a vlastní úloze) a zhodnoťte, jak vyhodnocovací aplikace zlepší vyhodnocení záznamu vůči stavu bez ní.

Seznam doporučené literatury:

[1] Michal Folta. Teaching of Transformations. Diplomová práce, FEL ČVUT, 2016. http://dcgi.fel.cvut.cz/theses/2016/foltamic
[2] Petr Felkel, Alejandra Magana, Michal Folta, Alexa Gabrielle Sears, Bedrich Benes I3T: Using Interactive Computer Graphics to Teach Geometric Transformations. Eurographics Education Papers 2018.
[3] Marek Nechanský, Automatic Box Layout in I3T Tool, Bachelor theses, Faculty of Nuclear Sciences and Physical Engineering, CTU Prague. 2019

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Felkel, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **13.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Petr Felkel, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Kapitola 1
Úvod

I3T je aplikace sloužící k výuce transformací ve 3D prostoru. Program umožňuje zobrazovat působení transformačních matic na tělesa a jejich vzájemné kombinování. Objekty zobrazené v 3D prostoru aplikace je nutné připojit na jednotlivé sekvence v pracovní ploše (workspace). Transformační matice se vkládají do sekvencí, čímž se aplikuje transformace. Sekvence mohou být také propojeny (connection), což má za výsledek násobení matic uvnitř sekvencí - ovlivněny jsou pouze sekvence, které mají příchozí spojení na svém vstupu. K propojení lze také využít různé operátory. Způsob propojení je stejný jako pro sekvence.

V aplikaci lze také animovat objekty pomocí operátoru float cycle, který cyklicky mění číselnou hodnotu. Tato hodnota se dá následně spojit s dalšími operátory pro vytvoření matice s měnícími se hodnotami.

I3T obsahuje klávesové zkratky pro ovládání pracovní plochy. Mezi hlavní zkratky patří:

- ctrl + levé tlačítko myši zkopíruje vybrané bloky na pracovní ploše
- klávesa “s” zobrazí všechny bloky na pracovní ploše
- klávesa “a” všechny bloky označí
- klávesa “i” invertuje označení
- klávesy “b” a “n” slouží k vracení operací (undo, redo)

Program obsahuje předpřipravené scény k procvičení různých témat týkajících se 3D transformací. Scény obsahují jednoduché návody k seznámení s ovládáním aplikace, jednoduché transformace objektů, ale také složitější scény, například pro vytvoření perspektivní kamery atp.

Poměrně velké množství funkcionalit může být pro uživatele matoucí. Proto je potřeba, aby aplikace I3T poskytovala uživatelům přívětivé a intuitivní grafické rozhraní. Pro nalezení chyb v grafickém rozhraní je nutno vytvořit v aplikaci I3T logovací systém uživatelské interakce. Pro snadnější vyhodnocení získaných záznamů o uživatelské interakci je potřeba vytvořit aplikaci, která je graficky zobrazí a umožní jejich vzájemné porovnávání. Součástí této práce bude vytvoření funkčního prototypu takovéto aplikace. Aplikace, jejímž hlavním účelem je prohlížení logů, se bude jmenovat Log Viewer.

Kapitola 2
Analýza

Kapitola se zabývá rešerší nástrojů pro sledování uživatelské interakce za pomoci nahrávání nebo sdílení obrazovky, dvou existujících nástrojů pro logování uživatelské interakce a možností vytvoření vlastího řešení. Dále se zabývá identifikací konkrétních potřeb pro logování I3T a vyhodnocovací aplikaci a v poslední řadě analýzou zpracování uživatelské interakce v programu I3T, pro integraci logovacího systému.

2.1 Rešerše nástrojů pro sledování uživatelské interakce

Nejjednoduší metodou sledování interakce uživatele s aplikací je přímé sledování uživatele, jako bychom mu „stáli za zády". Další metodou je získání textového záznamu z interakce. Sledování uživatelské interakce pomocí získání textového záznamu je rozšířené zejména pro webové a mobilní aplikace. Na těchto platformách můžeme nalézt poměrně velké množství nástrojů, které nám s tím dokáží pomoci. Pro desktopové aplikace bohužel tak rozsáhlé možnosti nemáme. Obliba sledování uživatelské interakce na těchto zmíněných platformách je pravděpodobně zapříčiněna tím, jak jednoduché je pro vývojáře webových a mobilních aplikací reagovat na získaná data, narozdíl od vývojářů pro desktop. Dostat k uživatelům aktualizaci desktopové aplikace není ve většině případů tak snadné, jako aktualizovat kód na serveru nebo vynutit aktualizaci mobilní aplikace v Google Play Store či App Store. Dalším důvodem, proč jsou první dvě platformy preferované, je garance získání nasbíraných dat. Většina nástrojů pro sledování uživatelské interakce ihned odesílá nasbíraná data přes internet. U většiny desktopových aplikací nelze zaručit připojení k internetu, tudíž riskujeme, že nasbíraná data nikdy nedostaneme.

Existuje velké množství nástrojů pro sledování uživatelské interakce (Google Analytics, Mixpanel, CrazyEgg, HotJar, Clicky, ClickTale, Flurry [1]). V následujících kapitolách bude podrobněji prozkoumána metoda nahrávání obrazovky a dva existující nástroje, které se zdály být nejvíce relevantní buď jako nástroje, které by bylo možné použít, nebo nástroje, ze kterých se můžeme inspirovat pro vytvoření vlastního řešení.

2.1.1 Sdílení a nahrávání obrazovky

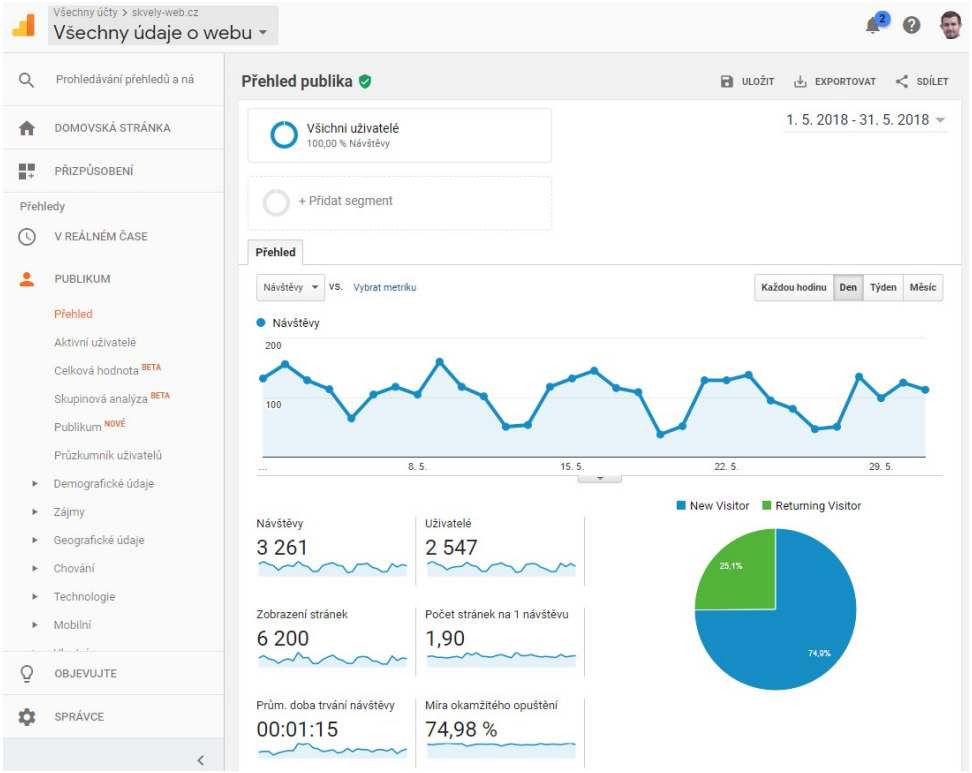
Pro sledování uživatelské interakce lze využít sdílení či nahrávání obrazovky. Sledování interakce tímto způsobem vyžaduje pozorovatele, který sleduje interakci s testovanou aplikací například pomocí funkcionality sdílení obrazovky v aplikaci Skype [6] s možností nahrání záznamu pro pozdější vyhodnocení. Pozorovatel může také zasahovat do interakce. Pokud není potřeba, aby pozorovatel zasahoval do interakce uživatele s testovanou aplikací, lze k nahrání využít například software CamStudio [7]. Nevýhodou těchto dvou přístupů je dlouhá doba nutná k vyhodnocení záznamu. Pro vyhodnocení je totiž potřeba zhlédnout celé video nahrané interakce pro každého uživatele zvlášť. Z tohoto důvodu je potřeba zvolit rychlejší metodu.

2.1.2 Google Analytics

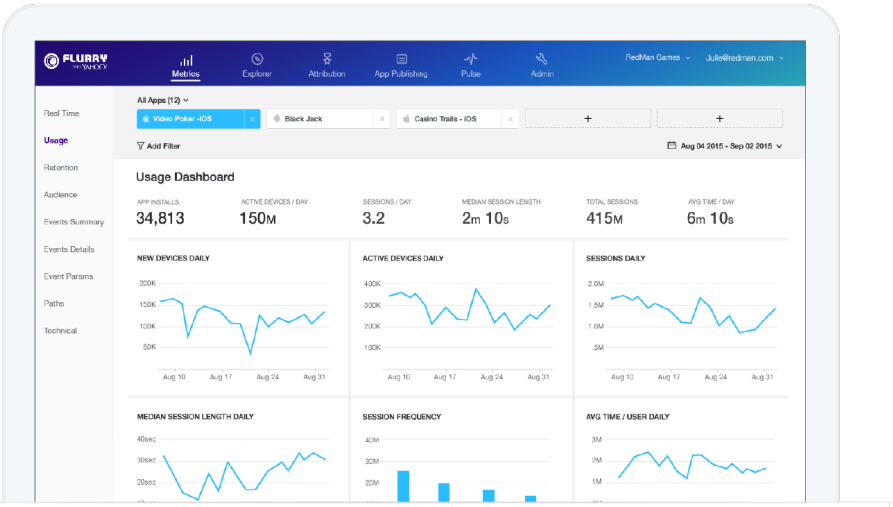
Jako zástupce sledovacích nástrojů pro webové aplikace byly zvoleny Google Analytics [2]. Google Analytics je nástroj vyvíjený a publikovaný společností Google. Jedná se o nejpoužívanější nástroj v této oblasti. Umožňuje získávat statistická data o uživatelích webových stránek. Mezi tato data patří například aktuální i historická návštěvnost (viz Obrázek 2.1), chování uživatelů a jejich vlastnosti (např. prohlížeč, operační systém, zařízení, rozlišení obrazovky, ale také demografické údaje jako země a město, ze kterého se uživatel připojuje a jazyk, který používá). Kromě toho umožňuje také tzv. sledování událostí (angl. Event tracking), které slouží k měření interakce s obsahem nezávisle na načítání webové stránky. Mezi tyto události může patřit například stažení souboru z webové stránky, spouštění videí, prokliky na odkazy a e-mailové kontakty atp. Přestože je nástroj určen pro webové aplikace, můžeme se jím inspirovat, hlavně využitím konceptu sledování událostí.

2.1.3 Flurry

Podobně jako Google Analytics je Flurry [4] zástupcem sledovacích nástrojů pro mobilní aplikace. Stejně jako Google Analytics umožňuje sledovat počet aktivních uživatelů aplikace (lze vidět na Obrázku 2.2) a získávat o nich statistická data, jako například operační systém, informace o zařízení, demografické informace o uživateli atp. Sledování událostí podporuje také. Flurry umožňuje vytvářet vlastní události, které lze parametrizovat podle požadavků. Parametrizace událostí u Flurry má více možností než události Google Analytics. Flurry je zdarma bez limitací.



Obrázek 2.1: Přehled získaných dat z Google Analytics [3]



Obrázek 2.2: Přehled získaných dat z Flurry [5]

2.1.4 Vlastní řešení

Při zkoumání existujících řešení se mi nepodařilo nalézt nástroj pro sledování uživatelské interakce určený pro desktopové aplikace. Kromě toho, že výše zmíněné nástroje cílí na jinou platformu, se také zabývají hlavně analýzou

uživatelů sledované aplikace z důvodu případného přizpůsobení aplikace různým cílovým skupinám. Hlavní cílová skupina aplikace I3T je daná, jedná se zejména o studenty FEL a FIT. Velká část funkcionality těchto nástrojů je pro nás tedy zbytečná.

Uvedené nástroje také počítají s tím, že má sledovaná aplikace přístup k internetu a všechna data posílají na server. Sledování uživatelské interakce v I3T bude probíhat v předem připravených testovacích podmínkách. Záznamy nám tedy stačí shromažďovat lokálně.

Kvůli tomu, že se musíme přizpůsobit struktuře aplikace I3T (podrobněji popsáno v Kapitole 2.3), potřebujeme nástroj, který je vytvořen na míru pro I3T. Z těchto důvodů jsem se rozhodl vytvořit vlastní logovací systém, který bude inspirován sledováním událostí, které využívají prozkoumané nástroje. Logovací systém bude součástí samotné aplikace I3T a bude mít definované API ve formě událostí, které budeme chtít logovat.

V následující kapitole budou prozkoumány detailní požadavky pro logování uživatelské interakce v I3T. Upřesnění těchto požadavků nám pomůže vytvořit ideální nástroj pro účely sledování uživatelské interakce.

2.2 Požadavky pro logování I3T

Protože bylo rozhodnuto, že nebude využit již existující nástroj pro sledování uživatelské interakce, ale bude vytvořen vlastní, budou nyní prozkoumány požadavky na toto sledování. Požadavky logování uživatelské interakce lze rozdělit na dvě části. První část se zabývá potřebami pro sledování z účelem vývoje grafického rozhraní aplikace. Pro docílení lepšího zpřístupnění nástrojů aplikace a intuitivního ovládání je potřeba mít způsob jak zjistit, zda uživatel v aplikaci zbytečně dlouho neváhá, nebo příliš dlouho po něčem nepátrá. Tyto informace podají zpětnou vazbu na nynější rozhraní a zároveň napoví, jak rozhraní vylepšit.

Druhá část obsahuje potřeby pro sledování řešení vzorových úloh v předpřipravených scénách. Informace o průběhu řešení úloh mohou být využity k optimalizaci připravených scén a úpravě výuky zaměřením na problematické části transformační teorie. V následujících kapitolách se podrobněji zaměříme na tyto dvě skupiny požadavků.

2.2.1 Vývoj rozhraní

Pro správný návrh a vývoj grafického rozhraní I3T je potřeba zjistit, které části stávajícího rozhraní jsou pro uživatele při práci problematické. Pro tyto účely je užitečné zaznamenávat *otevírání a zavírání vyskakovacích nabídek* a *klikání na jednotlivá tlačítka v těchto nabídkách*. Ke každému kliknutí je vhodné zaznamenat *čas*. Tyto informace lze následně porovnat s úkolem, který uživatel právě plní, a podle časových prodlev nebo „špatných kliknutí" nalézt problematické části. Dále potřebujeme periodicky zaznamenávat *pozici kurzoru*. Z těchto informací můžeme odhadnout, kde uživatel hledá mezi jednotlivými kliknutími.

Další část rozhraní, kterou můžeme sledovat, jsou *klávesové zkratky*. To je ovšem poměrně složité k evaluaci. Používání klávesových zkratek je velmi subjektivní pro každého uživatele. Většina programů implementuje různé klávesové zkratky. Záleží tedy, na co je konkrétní uživatel zvyklý při práci s jeho oblíbenými programy. Vyhodnocení tohoto sledování tedy pravděpodobně nebude příliš přínosné.

Požadavky na sledování řešení úloh se mírně liší od požadavků na sledování za účelem vývoje rozhraní. Tyto rozdíly budou popsány v následující kapitole.

2.2.2 Sledování řešení úloh

Sledování řešení vzorových úloh je nutné jak pro návrh samotných úloh, tak pro případnou optimalizaci seznámení s aplikací během výuky. Při sledování řešení úloh není natolik důležité sledovat jednotlivá kliknutí, ale spíše logické události v aplikaci. Potřebujeme sledovat *přidávání / odebírání komponent z pracovní plochy*, *vkládání matic do sekvencí* a *propojování sekvencí a operátorů*. Také lze využít sledování *změn hodnot v maticích*. Podle toho zjistíme, zda student nevyplňuje matice metodou pokus-omyl. Pomocí těchto informací můžeme vyhodnotit, zda student chápe náplň úloh a rozumí, jak transformace fungují.

Problémem při sledování logických událostí za účelem sledování řešení úloh může být špatně navržené rozhraní aplikace. Nepoznáme tedy, zda uživatel nechápe zadání úlohy, nebo pouze „bloudí" v nabídce a nemůže najít správné nástroje. Tento problém částečně vyřeší zapnutí sledování pro vývoj rozhraní (uvidíme, že prodleva je způsobena procházením různých částí nabídky), avšak za cenu zahlcení logu zbytečně detailními informacemi. Dále budou identifikovány konkrétní sledovací záměry pro jednotlivé kroky ve vzorových úlohách I3T.

2.2.3 Shrnutí požadavků na logování

Po identifikaci logovacích záměrů v Kapitolách 2.2.1 a 2.2.2 bylo zjištěno, že je potřeba vytvořit logovací systém, který podporuje různé úrovně logování, které se dají kombinovat a dynamicky měnit. Tyto úrovně jsou:

1. pohyb kurzoru
2. rozbalovací nabídky (otevírání, zavírání a klikání na tlačítka v nich)
3. zadané hodnoty v maticích
4. logické události v aplikaci

Dále musí logy obsahovat časový záznam. Přestože logy budou zpracovány strojem, chceme je také vyhodnocovat ručně. Proto musejí být jednoduché a přehledné i pro čtení a nesmí obsahovat zbytečné informace.

Protože je v plánu vytvořit logovací systém na míru pro potřeby I3T založený na sledování událostí (Event tracking), inspirovaný již existujícími nástroji pro sledování uživatelské interakce, je potřeba prozkoumat strukturu aplikace I3T, zejména zpracování uživatelských vstupů.

2.3 Struktura aplikace I3T

Po zjištění toho, které události je potřeba zaznamenávat, bylo dalším úkolem stanovit, kde v kódu aplikace I3T volat logovací API tak, aby byl kód udržitelný a přehledný. Bylo prozkoumáno, jak aplikace zpracovává uživatelské vstupy - hlavně klikání myší.

V aplikaci existuje bázová třída *Tab*, která představuje každý obdélníkový objekt (blok, matice, tlačítko, vstupní pole atp.) a ze které je následně děděno. Tato třída obsahuje funkce *mouseDown* a *mouseUp*, které zpracovávají stisknutá tlačítka myši. Tyto dvě funkce jsou zachovány pro většinu dědících tříd s výjimkou tříd *PopUpMenu* (rozbalovací nabídka), *CurveTab* (pole pro propojování sekvencí) a *NumberBox* (číselné pole v matici). Funkcionalita těchto procedur je popsána níže.

Každý *Tab* obsahuje seznam v něm vnořených *Tabů*, které jsou brány jako jeho potomci (např. pokud je matice uvnitř sekvence, bere se jako její potomek, sekvence je uvnitř pracovní plochy, takže je jejím potomkem atp.). Tato hierarchie je poté využívána při hloubkovém prohledávání potomků pro zpracování události kliknutí. Všechny *Taby* jsou také součástí jedné výchozí skupině *Tabů* *TabGroup*.

TabGroup obsahuje dvě vrstvy, ve kterých se nachází všechny *Taby*, a slouží ke zpracování vstupů z myši a z klávesnice pomocí své metody pro aktualizaci událostí (*updateEvents*). Při inicializaci programu jsou vytvořeny ve vrstvě 0 všechny operátory, rozbalovací nabídky i tlačítka, ale nejsou vykreslovány. Při jejich přidání do pracovní plochy je již vytvořeným objektům nastavena pozice podle kurzoru a začnou být vykreslovány. Ve vrstvě 1 je vytvořena pracovní plocha a hlavní menu v horní části obrazovky.

Tab také obsahuje ukazatele na funkce, které slouží jako události při kliknutí (*onMouseDown, onMouseUp, onPassiveMouseUp*). Tyto funkce jsou implicitně nastaveny na NULL a pouze některé dědící třídy je mají nadefinované. Někdy jsou tyto funkce nastaveny zvlášť až po inicializaci objektu, např. funkcionalita jednotlivých tlačítek. V následujících kapitolách bude podrobně popsán způsob zpracování kliknutí myši.

2.3.1 Zpracování kliknutí myši

Po prozkoumání hierarchie tříd v aplikaci bylo zjištěno, jak konkrétně probíhá zpracování kliknutí myši a jak se aktivují různé události v aplikaci. Kliknutí sestává ze dvou částí - stlačení a uvolnění tlačítka myši. Každá tato část je zpracována samostatně.

Pokud bylo stisknuto tlačítko myši, funkce *updateEvents* v *TabGroup* iterativně volá *mouseDown* postupně pro všechny *Taby* v obou vrstvách v pořadí vrstva 0, vrstva 1, jak bude popsáno dále. Každý *Tab* přitom rekurzivně prohledá své potomky a volá jejich metodu *mouseDown*, která pomocí pozice kurzoru určí, zda bylo kliknuto právě na něho. Tato kombinace iterativního a rekurzivního prohledávání probíhá do té doby, než je nalezen nejhlubší kliknutý *Tab*. Pro tento nejhlubší kliknutý *Tab* je nastavena hodnota proměnné

„clicked", která je potřeba pro volání událostí při zpracování uvolnění tlačítka myši.

Stejným způsobem je zpracována událost uvolnění tlačítka myši, s tím rozdílem, že je volána funkce *mouseUp*. Tato funkce stejným způsobem prohledává své potomky, dokud nenalezne nejhlubší *Tab*, na kterém bylo tlačítko myši uvolněno. Pro nalezený nejhlubší kliknutý *Tab* jsou zavolány události kliknutí. Funkce *mouseUp* je v principu stejná pro všechny druhy *Tabů*. Podstatně se odlišuje třída jen *PopUpMenu*. Funkce *mouseUp* této třídy při vyhodnocení, že nebylo na *Tab* kliknuto, nevrátí pouze hodnotu NULL, jako ostatní *Taby*, ale před tím ještě zavře rozbalené *PopUpMenu*.

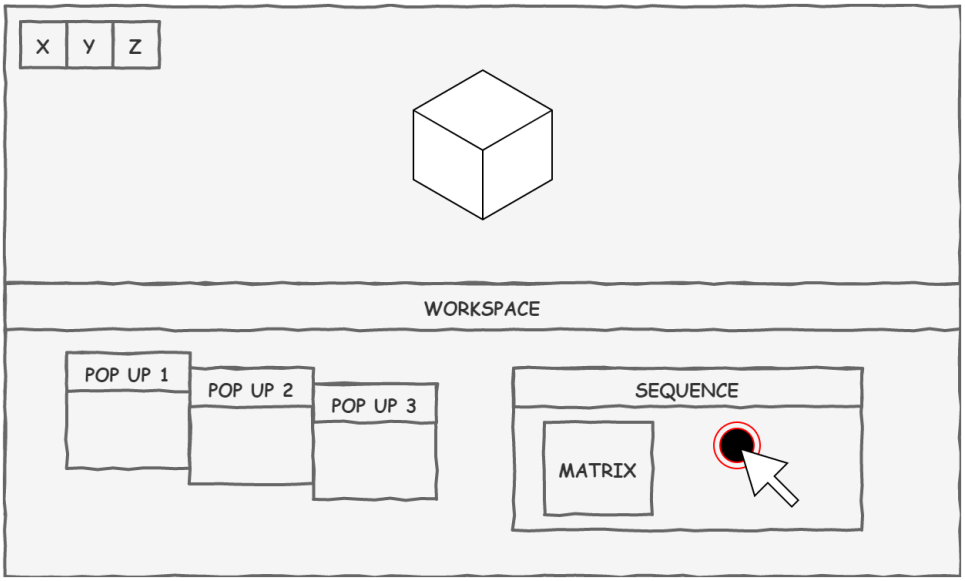
Taby jsou v *TabGroup* rozděleny do dvou vrstev kvůli pořadí jejich prohledávání. Nejdříve jsou prohledány *Taby* ve vrstvě 0. Vrstva 0 obsahuje všechny rozbalovací nabídky. Rozbalovací nabídky jsou prohledány jako první kvůli jejich případnému zavření ve funkci *mouseUp*. Poté jsou prohledány *Taby* ve vrstvě 1. Tzn. tlačítka v hlavním menu a pracovní plocha (společně s bloky, které jsou na ni rozmístěné, jako jejími potomky). Provolávání funkce *mouseUp* je podrobně ukázáno na dvou příkladech v následující kapitole.

2.3.2 Příklady zpracování kliknutí

Postupy zpracování události kliknutí myši z předchozí kapitoly ilustrujeme na dvou příkladech zpracování uvolnění tlačítka myši na stejném místě, na kterém bylo stisknuto. Příklady jsou zobrazeny pomocí jednoduchých obrázků s naznačením provolávaných funkcí.

Pozn.: Vysvětlení notace obrázků: volání funkcí probíhá v pořadí odshora dolů, odsazení řádku za dvojtečkou znamená vnoření do funkce před dvojtečkou, „->" značí důsledek vyhodnocení podmínky před znakem nebo na řádcích před znakem, „{layer x}" značí právě testovanou vrstvu

Při kliknutí do sekvence, které je naznačeno na Obrázku 2.3, dochází k volání funkce *mouseUp* třídy *TabGroup*. Ta volá pomocí dvou vnořených cyklů *mouseUp* všech *Tabů* v obou vrstvách *TabGroup*. *Taby* ve vrstvě 0 jsou všechny rozbalovací nabídky. Nabídky, které nejsou viditelné (rozbalovací nabídky sekvencí, operátorů atp.) vrací NULL. Viditelné nabídky (v obrázku zobrazeny jako POP UP 1, 2, 3 - rozbalovací nabídka pracovní plochy) vrací NULL, protože pozice kurzoru nesouhlasí s jejich pozicí ani s pozicí jejich potomků a jsou zavřeny (*setVisible(false)*). Dále je prohledávána vrstva 1. Ve vrstvě 1 se nachází hlavní menu (v obrázku naznačeno tlačítky X, Y, Z) a pracovní plocha (v obrázku WORKSPACE). Tlačítka v hlavním menu vrací NULL, protože pozice kurzoru nesouhlasí s jejich pozicí. Při volání *mouseUp* pracovní plochy je zjištěno, že jeden z jejích potomků (v obrázku SEQUENCE, do které bylo kliknuto), zpracoval událost kliknutí tím, že zavolal své události (*onMouseUp, onPassiveMouseUp*) a vrátil sám sebe. Pracovní plocha vrátí potomka, který kliknutí zpracoval do funkce *mouseUp* třídy *TabGroup*, čímž je indikováno, že došlo ke zpracování události a dvojitý vnořený cyklus se ukončuje.

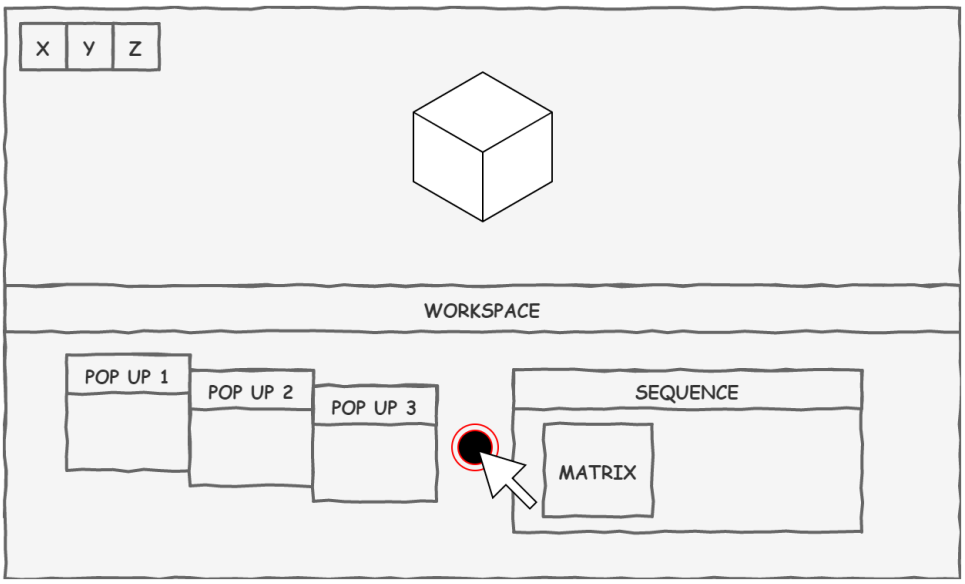



```
TabGroup::mouseUp:
  {layer 0}
  other PopUpMenus: PopUpMenu::mouseUp:
    !visible -> return NULL
  POP UP 1,2,3: PopUpMenu::mouseUp:
    children::mouseUp returned NULL
    !mouseIn -> setVisible(false)
    -> return NULL
  {layer 1}
  X,Y,Z: Tab::mouseUp:
    children::mouseUp returned NULL
    !mouseIn
    -> return NULL
  WORKSPACE: Tab::mouseUp:
    SEQUENCE: Tab::mouseUp:
      children::mouseUp returned NULL
      mouseIn -> onPassiveMouseUp()
      clicked -> onMouseUp()
      -> return this
    children::mouseUp returned SEQUENCE
    -> return SEQUENCE
```

Obrázek 2.3: Pořadí testování jednotlivých objektů rozhraní po uvolnění tlačítka myši na sekvenci

Při kliknutí do pracovní plochy (mimo menu a bloky), které je naznačeno na Obrázku 2.4, probíhá začátek zpracování stejně jako v předchozím příkladu. Pracovní plocha však zjistí, že nebylo kliknuto na žádného jejího potomka (všichni vrátili NULL), a proto volá svou událost a vrací sama sebe, čímž

také ukončuje vyhledávání v *TabGroup*.




```
TabGroup::mouseUp:
  {layer 0}
  other PopUpMenus: PopUpMenu::mouseUp:
    !visible -> return NULL
  POP UP 1,2,3: PopUpMenu::mouseUp:
    children::mouseUp returned NULL
    !mouseIn -> setVisible(false)
    -> return NULL
  {layer 1}
  X,Y,Z: Tab::mouseUp:
    children::mouseUp returned NULL
    !mouseIn
    -> return NULL
  WORKSPACE: Tab::mouseUp:
    SEQUENCE: Tab::mouseUp:
      children::mouseUp returned NULL
      !mouseIn
      -> return NULL
    children::mouseUp returned NULL
    mouseIn -> onPassiveMouseUp()
    clicked -> onMouseUp()
    -> return this
```

Obrázek 2.4: Pořadí testování jednotlivých objektů rozhraní po uvolnění tlačítka myši na pracovní ploše (mimo menu a bloky)

2.3.3 Události kliknutí myši

Události nejsou definovány pro všechny *Taby*. Může se také stát, že *Tab*, který kliknutí zpracovává (tzn. je nejhlubší kliknutý *Tab*), nemá definovanou událost při kliknutí (nemá vůbec žádnou, nebo pouze pro pravé tlačítko, ale bylo kliknuto levým - například pracovní plocha má na pravém tlačítku otevření rozbaovací nabídky, na levém nic). V tomto případě se neděje nic, *Tab* je pouze navrácen a *TabGroup* ukončuje hledání.

Kvůli tomu, že implementace jednotlivých událostí jsou rozmístěny v různých třídách, se kterými události souvisí (např. zobrazení rozbalovací nabídky s možnostmi pro sekvenci se nachází ve třídě sekvence), nejsou funkce *onMouseUp* a *onPassiveMouseUp* vhodným místem pro logování událostí, protože by logování bylo „rozházené" v různých částech kódu.

Taby rozlišují události *onMouseUp* a *onPassiveMouseUp*. Funkce *onMouseUp* je volána pouze tehdy, pokud bylo na *Tabu*, na kterém bylo tlačítko myši uvolněno, tlačítko myši také stisknuto. Aby se dalo zjistit, zda uživatel uvolnil tlačítko na stejném *Tabu*, jako na kterém ho stiskl, má každý *Tab* proměnnou typu boolean „clicked", která je při každém zpracování stisknutí tlačítka myši aktualizována. Událost *onPassiveMouseUp* se volá pokaždé, když je tlačítko na *Tabu* uvolněno (kromě klasického kliknutí na *Tab* také ukončení tažení (*drag*) odjinud). Slovo *passive* v názvu funkce je zavádějící, protože slovo *passive* se obvykle používá pro sledování pohybu bez stisknutých tlačítek, a proto působí jako obsluha situace, kdy nebylo kliknuto vůbec. Názvy těchto metod by proto měly být změněny. Funkce *onMouseUp* by se mohla jmenovat například *onMouseUpOnClickedTab*, protože je volána pouze v případě, že bylo na *Tabu* tlačítko také stisknuto. Funkce *onPassiveMouseUp* by se mohla jmenovat *onMouseUp*, protože je volána při každém uvolnění tlačítka myši na *Tabu*. Seznam všech událostí v aplikaci se nachází v Příloze B.

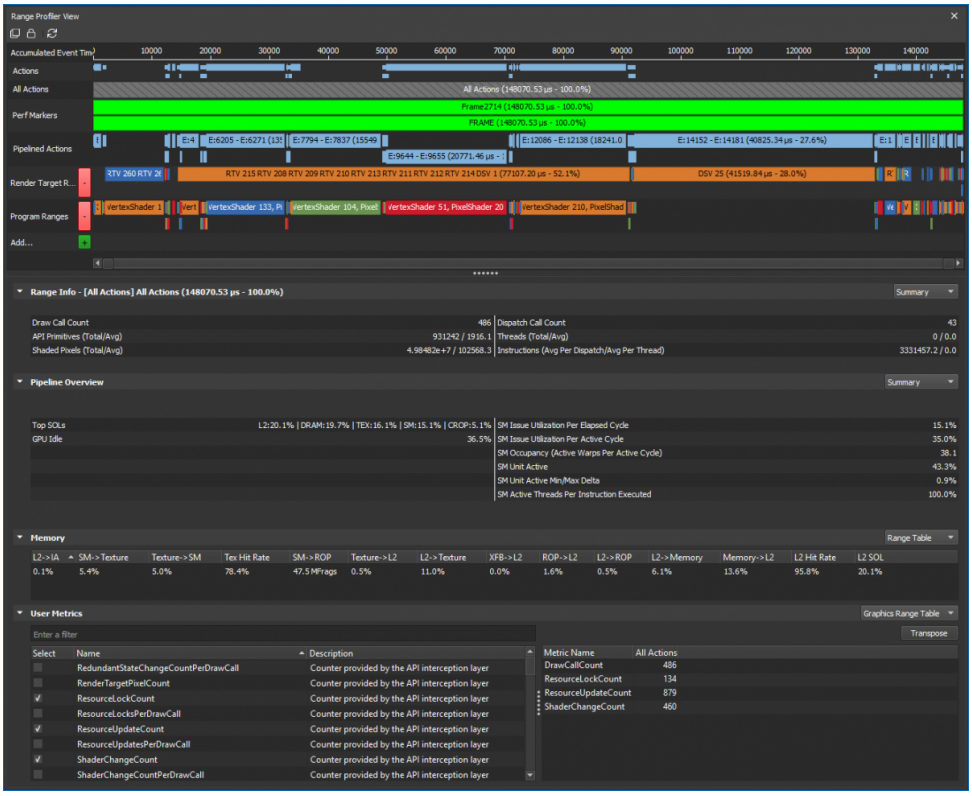
Po získání hrubé představy o tom, co bude logováno, můžeme začít přemýšlet o tom, jakým způsobem usnadnit vyhodnocování získaných záznamů. Tímto se bude zabývat aplikace Log Viewer. Požadavky na tuto aplikaci budou stanoveny v následující kapitole.

2.4 Funkční požadavky Log Viewer

Po stanovení požadavků na logování uživatelské interakce máme hrubou představu o tom, co vznikne za záznamy. Druhou částí bakalářské práce je vytvoření funkčního prototypu aplikace Log Viewer, který usnadní vyhodnocování získaných logů. Tato kapitola se bude zabývat analýzou funkčních požadavků pro tuto aplikaci.

Pro snadné vyhodnocení logu potřebujeme převést textový záznam uživatelské interakce do grafické podoby. Inspirací pro možný vzhled vyhodnocovací aplikace je NVIDIA Nsight debugger [8], který lze vidět na Obrázku 2.5. NVIDIA Nsight slouží k ladění aplikací, které využívají různé grafické knihovny, včetně OpenGL. Zobrazuje v několika časových osách různé činnosti grafické karty. Časové osy vykresluje paralelně pod sebou a tím zobrazuje všechny

probíhající události v časové návaznosti.



Obrázek 2.5: Ukázka obrazovky NVIDIA Nsight [8]

Pro účely přehledného zobrazení zaznamenaných událostí v aplikaci bude využit tento koncept časových os. Události budou rozděleny na dlouhotrvající a jednorázové akce, které budou zobrazeny v časových osách pod sebou. Dlouhotrvající akce jako barevný obdélník, jednorázové akce jako svislá čára. Časové osy budou pro lepší přehlednost různě barevné na základě toho, čeho se záznam týká (např. práce s rotační maticí bude mít jinou barvu než práce s translační maticí). Časové osy půjdou přibližovat a oddalovat kolečkem myši.

Pro porovnání uživatelů potřebujeme zobrazit více logů najednou. Pro jednodušší manipulaci s porovnávanými logy potřebujeme implementovat synchronizaci časových os. Synchronizace bude probíhat na základě záznamů o plnění tutoriálových kroků. Tutoriály prozatím nejsou součástí aplikace I3T. Proto bude v logovacím systému tato událost připravena a její zalogování bude pro testovací účely simulováno stiskem klávesy.

Při vyhodnocování logu uživatele je vhodné zobrazit tzv. zlatý průchod testovaným scénářem paralelně s vyhodnocovaným logem. Tento zlatý průchod bude sloužit jako minimální příklad toho, co měl uživatel udělat, aby splnil daný úkol. Scénář bude obsahovat stejné informace jako běžný záznam uživatelské interakce. Proto bude také zobrazen stejným způsobem pomocí časových os.

Po identifikaci požadavků na logování, prozkoumání struktury aplikace

I3T a analýze funkčních požadavků pro Log Viewer můžeme začít řešit návrh implementace logovacího systému pro I3T a zmíněné vyhodnocovací aplikace. Tímto návrhem se bude zabývat následující kapitola.

Kapitola 3
Návrh

V předchozí kapitole byla provedena analýza požadavků na logování uživatelské interakce a byly stanoveny funkční požadavky pro vyhodnocovací aplikaci Log Viewer. Na základě těchto požadavků je potřeba zvolit technologie, které budou využity pro implementaci logovacího systému a vyhodnocovací aplikace. V kapitole se budeme nejprve zabývat výběrem logovací knihovny pro logovací systém I3T. Dále bude navržen diagram tříd aplikace Log Viewer a zvolen programovací jazyk, kterým bude aplikace napsána. Na závěr se budeme věnovat návrhu funkčních prvků vyhodnocovací aplikace.

3.1 Výběr logovací knihovny

Pro vytvoření logovacího systému je nutno vybrat vhodnou logovací knihovnu, která pomůže naplnit požadavky logovacího systému definované v předchozí Kapitole 2.2, a nebo vytvořit vlastní. Pro jazyk C++ existuje poměrně velké množství kvalitních logovacích knihoven, tudíž psaní vlastní knihovny není nutné. Tato možnost byla tedy zavrhnuta již na začátku.

Vzhledem k tomu, že je plánována podpora Linuxu, musí být vybraná knihovna cross-platformová. Na internetu byly nalezeny tři knihovny, které se zdály splňovat požadavky. Pro ně byla vytvořena testovací implementace v projektu I3T, pro podrobnější seznámení. Při testování logovacích knihoven vznikla kritéria pro výběr knihovny, která vychází z požadavků analyzovaných v Kapitole 2.2. Nejdůležitější kritérium je cross-platformová podpora. Pro splnění požadavku na přehledné záznamy je nutné, aby bylo možné upravit formát logu a rozdělit log do více souborů. Nejméně důležitý je požadavek na funkcionalitu podmíněného logování, z důvodu jednoduchosti jeho dodatečné implementace.

3.1.1 Loguru

Knihovna Loguru [9] obsahuje již implementované podmíněné logování, které by se hodilo pro splnění požadavku dynamicky měnitelných úrovní logování. Je také cross-platformová. Po krátkém testování bylo zjištěno, že log obsahuje informace, které nejsou pro sledování uživatelské interakce důležité (např. vlákno a soubor, ze kterého je log volán) a formát logu nelze žádným způsobem

modifikovat. Knihovna se tedy hodí spíše pro logování za účelem debugování než pro sledování uživatelské interakce.

3.1.2 Easylogging++

Další knihovna, která byla vyzkoušena, byla Easylogging++ [10]. Součástí knihovny je podmíněné logování a knihovna je také cross-platformová. Formát logu lze jednoduše nastavit pomocí konfiguračního souboru tak, aby obsahoval jen ty informace, které jsou pro nás důležité, tzn. pouze čas a zpráva.

Po implementování větší části funkcionality bylo zjištěno, že při periodickém logování pozice kurzoru začíná být log velmi nepřehledný. Log bylo tedy třeba rozdělit do dvou výstupových souborů. Jeden soubor pro pozici kurzoru, druhý soubor pro zbytek logů. Bohužel tato knihovna nepodporuje rozdělení logů do různých souborů, ani nelze instanciovat více různých loggerů s odlišným nastavením pro ukládání.

3.1.3 Spdlog

Poslední knihovnou, která byla vyzkoušena, byl Spdlog [11]. Knihovna je cross-platformová, formát logu lze upravit tak, jak potřebujeme, a je zde možnost instanciovat více loggerů se zapisováním do různých souborů. Tato knihovna bohužel neobsahuje podmíněné logování, ale tento nedostatek není obtížné implementovat.

3.1.4 Zhodnocení knihoven

Žádná z testovaných knihoven nesplňuje všechny požadavky, je tedy nutné při výběru zohlednit prioritu jednotlivých kritérií. Knihovna Spdlog splňuje důležitější požadavky. Z tohoto důvodu byla vybrána.

Přehled logovacích knihoven a požadavků				
	Cross-platform	Úprava formátu logu	Rozdělení do více souborů	Podmíněné logování
Loguru	ANO	NE	ANO	ANO
Easy-logging++	ANO	ANO	NE	ANO
Spdlog	ANO	ANO	ANO	NE

Tabulka 3.1: Tabulka požadavků a porovnávaných knihoven - sloupce požadavků jsou seřazeny podle priority (vyšší priorita vlevo)

3.2 Programovací jazyk pro Log Viewer

Když byla vybrána vhodná logovací knihovna pro logovací systém I3T, je na řadě zvolit technologie k tvorbě vyhodnocovací aplikace. Aplikace Log

Viewer bude psána v programovacím jazyce Python [12], verze 3.8. Python je interpretovaný skriptovací programovací jazyk. Jedná se o hybridní jazyk, což znamená, že program nemusí být celý objektově orientovaný, ale části programu mohou mít procedurální charakter. To přispívá k lepší čitelnosti kódu a celkovému zjednodušení. Python byl vybrán hlavně kvůli jeho jednoduchosti a předchozím zkušenostem programátora s tímto jazykem.

Pro Python existuje velké množství dostupných knihoven, které lze jednoduše spravovat pomocí správce balíků *pip* [13] a virtuálního prostředí *venv* [14]. Mezi standardní knihovny jazyka Python patří také knihovna *TkInter* [15]. Více o těchto tématech bude objasněno v následujících kapitolách.

3.2.1 TkInter

TkInter je knihovna pro tvorbu grafických rozhraní v jazyce Python, která je součástí standardní instalace Pythonu. Knihovna je multiplatformní. Díky tomu bude program fungovat stejně na operačním systému Windows i Linux.

Součástí knihovny *TkInter* jsou tzv. widgety pro nejběžnější prvky grafických rozhraní. Mezi tyto patří například okno, tlačítko, nadpis, rozbalovací nabídka atp. Využití těchto widgetů velmi usnadní práci při tvorbě grafického rozhraní Log Vieweru.

Widgety *TkInter* existují ve dvou variantách. První varianta jsou plně přizpůsobitelné widgety z výchozího balíku *TkInter*. Druhou variantou jsou tzv. *ttk* (themed tk) widgety [16]. Vzhled *ttk* widgetů lze spravovat jednotně pomocí stylů. Knihovna *TkInter* obsahuje sedm základních stylů a další lze importovat například z knihovny *ttkthemes* [17]. Widgety *ttk* a styly z knihovny *ttkthemes* budou využity pro další usnadnění práce při tvorbě grafického rozhraní.

Příklad inicializace programu využívajícího *TkInter* a aplikace jednotného stylu lze vidět ve výřezu kódu (tzv. snippetu) na Obrázku 3.1.

```
from ttkthemes import themed_tk as tk

WINDOW_TITLE = "My app"
WINDOW_SIZE = "1440x900"

root = tk.ThemedTk()
root.title(WINDOW_TITLE)
root.set_theme("black")
root.geometry(WINDOW_SIZE)
root.mainloop()
```

Obrázek 3.1: Inicializace grafického rozhraní *TkInter* a aplikace *ttk* stylu

3.2.2 Virtuální prostředí venv

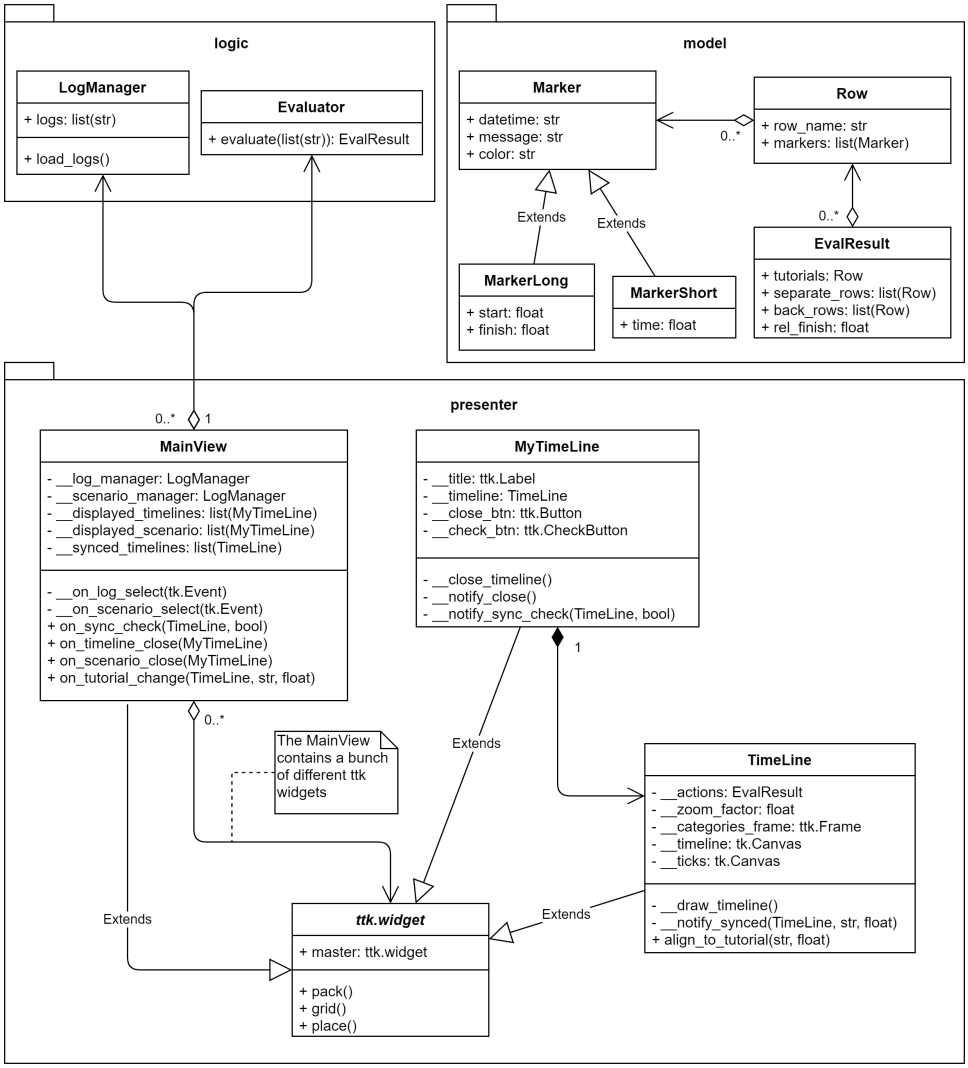
V předchozí kapitole byla zmíněna knihovna *TkInter*, která je součástí standardní instalace Pythonu. Kromě této knihovny využijeme i knihovnu *ttkwidgets*, kterou se samotným Pythonem nezískáme. Proto je potřeba se seznámit s „Pythonovskými" nástoji pro správu externích knihoven. Virtuální prostředí *venv* umožňuje oddělit systémovou instalaci Pythonu od konkrétního projektu tím, že vytvoří kopii instalace Pythonu včetně *pipu* přímo v adresáři s naším projektem. Spolu s touto kopií instalace vytvoří také skript pro aktivaci nového prostředí, který nastaví dočasné systémové proměnné na zkopírovanou instalaci. Díky tomu nemusíme instalovat další knihovny globálně a vyhneme se problémům, které přicházejí se závislostmi na různých verzích knihoven mezi projekty.

Venv je balík, který lze klasicky stáhnout pomocí správce balíků pip. Je to teoreticky jediný balík, který by měl být nainstalován do systémové instalace Pythonu. Všechny ostatní knihovny je lepší instalovat do virtuálních prostředí jednotlivých projektů, abychom předcházeli zbytečným problémům. [18] Když víme, jaké technologie pro vývoj aplikace Log Viewer použijeme, můžeme se začít zabývat jejím samotným návrhem. V následující kapitole bude budou navrženy třídy a jejich relace, které se budou vyskytovat v aplikaci.

3.3 Diagram tříd aplikace Log Viewer

Po zvolení technologií pro vývoj postoupíme k návrhu struktury aplikace. Pro tento účel byl vytvořen diagram tříd pomocí modelovacího jazyka UML [19]. UML je grafický jazyk, který lze použít také pro vizualizaci a navrhování programových systémů. V diagramu tříd jsou zobrazeny jednotlivé třídy, které se nachází v aplikaci, společně s jejich atributy a datovými strukturami těchto atributů, metodami tříd a souvislostmi mezi objekty.

Z diagramu tříd na Obrázku 3.2 je patrné, že aplikace bude sestávat ze tří hlavních částí. Mezi tyto části patří balík *logic*, který se stará o logickou část aplikace, tzn. nahrávání logů ze souborů a jejich zpracování. Dalším balíkem je *model*. Třídy v tomto balíku slouží jako obal pro data, která se předávají uvnitř aplikace. Balík *model* slouží hlavně k zajištění lepší organizace a čitelnosti kódu. Posledním balíkem je *presenter*, který obsahuje všechny widgety z knihovny *TkInter* použité v tomto projektu, zejména pak vlastní widget pro časovou osu, který je z velké části inspirován kódem časové osy z knihovny *ttkwidgets* [20]. Časová osa z této knihovny nevyhovovala našim požadavkům hlavně kvůli nemožnosti synchronizace více časových os. Protože je ale knihovna vyvíjena pod licencí GNU General Public License [21], můžeme její kód převzít a upravit podle vlastních požadavků. Součástí balíku je také třída *MainView*. Tato třída obsahuje všechny zobrazené widgety a stará se o interakci s uživatelem (vybírání logů, zobrazování / zavírání časových os, synchronizace os atp.).



Obrázek 3.2: Diagram tříd aplikace Log Viewer

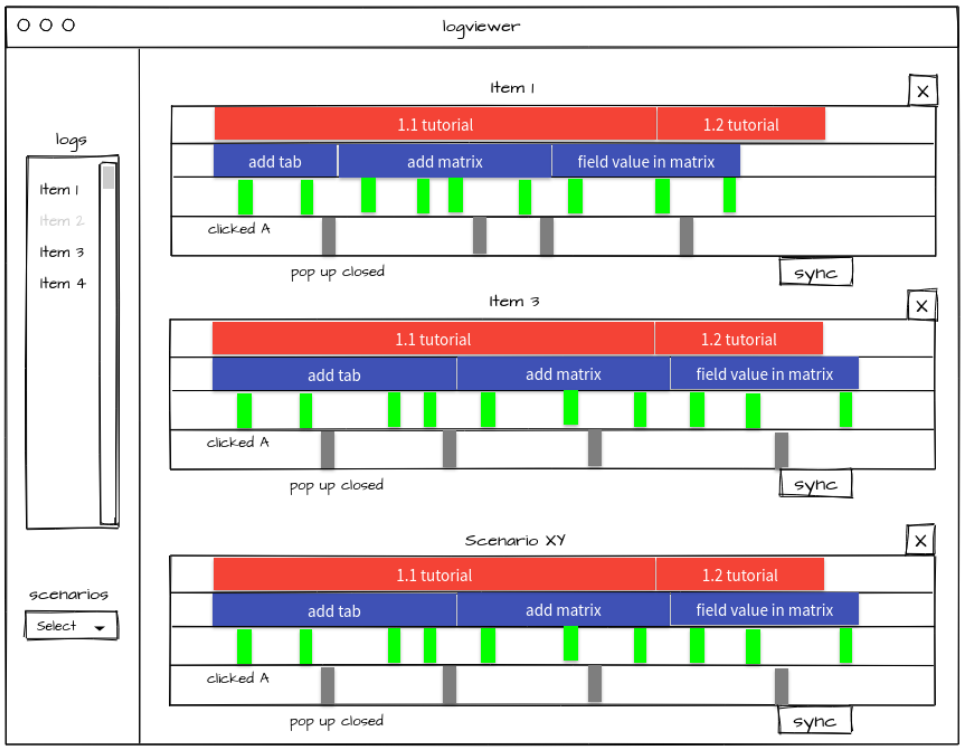
3.4 Návrh rozhraní aplikace Log Viewer

Když máme představu o objektové struktuře programu, můžeme se nyní začít věnovat návrhu rozhraní apliakce. Za tímto účelem byl vytvořen wireframe [22]. Wireframe na Obrázku 3.3 obsahuje rozvržení funkčních prvků na obrazovce. Jelikož cílem práce je vytvořit funkční prototyp aplikace, nebudeme se příliš zabývat grafickým návrhem aplikace. Pro *ttk* widgety bude aplikován styl „black" z knihovny *ttkthemes* (viz Kapitola 3.2.1), který se podobá vzhledu aplikace I3T.

Aplikace je navržena jako jedna obrazovka obsahující dvě hlavní části - menu a časové osy. V levé části obrazovky se nachází menu. V horní části menu je seznam nalezených záznamů. V seznamu je zobrazeno více záznamů, protože očekáváme, že uživatel bude mít k dispozici větší množství záznamů, a také proto, že jich může zvolit více najednou. Ve spodní části menu se

nachází rozbalovací nabídka s nalezenými vzorovými scénáři. Rozbalovací nabídka zabírá méně místa na obrazovce a aplikace nepodporuje zobrazení více scénářů současně.

V pravé části obrazovky je prostor pro grafické zobrazení logů a scénářů. Záznam uživatelské interakce je vyobrazen pomocí čtyřech časových os. První časová osa obsahuje záznam o právě probíhajícím kroku tutoriálu. Druhá časová osa znázorňuje, jakou logickou událost v aplikaci právě uživatel vykonává. Ve třetí ose jsou znázorněna jednotlivá kliknutí na tlačítka v otevřených rozbalovacích nabídkách a poslední osa zobrazuje zavření rozbalovacích nabídek.



Obrázek 3.3: Wireframe aplikace Log Viewer

Kapitola 4
Implementace

V předchozí kapitole byly zvoleny technologie pro implementaci logovacího systému a vyhodnocovací aplikace. Na základě toho může být provedena implementace. Kapitola obsahuje popis způsobu implementace logovacího systému a jeho integrace do aplikace I3T na základě průzkumu v Kapitole 2.3. Dále se v kapitole budeme zabývat popisem implementace aplikace pro vyhodnocování logů Log Viewer.

4.1 Popis implementace loggeru

Jako první je potřeba implementovat logovací systém do aplikace I3T a vystavit API k logování eventů pro zbytek aplikace. Za tímto účelem byla vytvořena třída *Logger*. Třída má definováno API pomocí maker kvůli budoucímu rozlišení sledovaných a nesledovaných verzí aplikace. Formát zpráv logovaných v jednotlivých makrech je uložen v souboru *events_definition.json* (viz Příloha C), který se nachází v adresáři s aplikací I3T.

Uvnitř třídy je realizováno rozdělení a kontrola úrovní. Úrovně logování jsou rozděleny podle požadavků na:

1. periodické logování pozice kurzoru (*MouseRaw*)
2. logování rozbalovacích nabídek (*PopUps*)
3. logování logických událostí (*Logic*)
4. logování polí v maticích (*MatrixFields*)

Pro každou úroveň obsahuje *Logger* proměnnou typu *boolean*, kterou lze dynamicky přepínat pomocí kláves F1 až F4. Na základě těchto proměnných je logování určitých částí vypnuté či zapnuté. Úrovně lze libovolně kombinovat.

Pro zapisování logů do souborů využívá třída knihovnu *Spdlog*, která byla vybrána v předchozí Kapitole 3.1. Obsahuje dvě, resp. tři instance knihovního loggeru. Jednu pro periodické logování myši a druhou pro ostatní záznamy, kvůli lepší přehlednosti logů. V případě, že se jedná o verzi sestavenou v debug modu, obsahuje třída ještě třetí logger, který vypisuje záznamy do konzole pro usnadnění vývoje logovacího systému.

Logger obsahuje také paměť pro uchování rozpracovaných událostí. Zprávy, kterí se do ní přidávají pomocí speciálního makra, se do logu zapíší až při ukončení celé události zavoláním další funkce pro logování. Do logu se tak zapíše celá tato paměť společně s nvou zprávou pod jedním časovým záznamem. Tato paměť je využívána například při logování propojení uzlů v pracovní ploše. Pokud je pokus o spojení neúspěšný, zaznamenáme pokus spolu se zprávou o neúspěchu pod jedním časovým záznamem.

Definovaná makra pro logování reprezentují jednotlivé události v aplikaci a při těchto akcích jsou volána. Některá makra očekávají jako vstupní parametry ukazatele na *Tab*, jejichž identifikátory jsou použity pro formátování logované zprávy. V některých makrech probíhá také dodatečná kontrola, například že logování stisknutí tlačítka v rozbalovací nabídce, které je v kódu voláno ze základní třídy *Tab*, je zaznamenáno pouze pro instance typu *Button* a pro ostatní typy instancí (například *TransformationForm - sekvence*) je ignorováno.

Využívaná logovací knihovna sice podporuje formátování zpráv pomocí většího množství argumentů, ale kvůli přidání potřebné funkcionality logovací paměti a pro filtrování záznamů je tato funkčnost rozšířena a implementována znovu. Metody pro logování a pro přidání zpráv do paměti jsou proto variadické funkce (funkce s různým počtem parametrů). V těchto funkcích probíhá kontrola typu logu a kontrola časovače logování pohybu kurzoru (pozice kurzoru je logována každých x sekund). Vlastní formátovací funkce je poté připravena na vstupní parametry typu string, pomocí nichž naformátuje zprávu pro záznam do logu.

4.2 Popis API logovacího systému

V hlavičkovém souboru třídy *Logger* jsou definována makra, která slouží jako API logovacího systému. Aby bylo zajištěno, že bude otevřen pouze jeden logovací soubor, je třída *Logger* implementována jako singleton, proto každé makro volá metodu pro zapsání do logu na jedné instanci třídy *Logger*. *Logger* obsahuje sedmnáct maker pro logování událostí v aplikaci. Logovací makra lze rozdělit do čtyřech kategorií podle logovacích úrovní. Dále existuje makro pro přidání do logovací paměti (popsáno v předchozí kapitole) a makra pro inicializaci, ukončení a aktualizaci *Loggeru*.

Logování pozice kurzoru

Makra pro logování pozice kurzoru očekávají na vstupu souřadnici x a y pozice kurzoru na obrazovce. Makra, která zaznamenávají pozici při kliknutí také potřebují typ tlačítka. Všechny tyto argumenty musejí být dodány jako *string*. Ukázka maker pro logování pozice kurzoru je na Obrázku 4.1.

```
# define LOG_EVENT_MOUSE_POS(mouseX, mouseY)
# define LOG_EVENT_MOUSE_CLICK(button, mouseX, mouseY)
# define LOG_EVENT_MOUSE_RELEASE(button, mouseX, mouseY)
```

Obrázek 4.1: Makra pro periodické logování pozice kurzoru a pozice kurzoru při stlačení a uvolnění tlačítka myši

Logování rozbalovacích nabídek

Makra pro logování otevírání, zavírání a klikání na tlačítka v rozbalovacích nabídkách jsou na Obrázku 4.2. Všechna tato makra potřebují argument typu *Tab*. V případě otevření nabídky tento argument reprezentuje *Tab*, na který bylo kliknuto, a tudíž je jeho nabídka otevřena (pracovní plocha, sekvence, matice atp.). V případě zavření nabídky se jedná o *Tab* zavřené nabídky (tzn. instance třídy *PopUpMenu*). Při zavírání nabídky rozlišujeme zprávu o zavření z důvodu kliknutí mimo nabídku (*LOG_EVENT_CLOSE_POP_UP*) a zavření z důvodu kliknutí do nabídky (*LOG_EVENT_CLOSE_POP_UP_IN*). Zavření z důvodu kliknutí do nabídky nastává při kliknutí na tzv. „poslední" tlačítko v nabídce (např. tlačítko, které přidá sekvenci zároveň zavře nabídku). Makro pro logování kliknutí na tlačítko očekává jako argument tlačítko, na které bylo kliknuto.

```
# define LOG_EVENT_OPEN_POP_UP(clickedTab)
# define LOG_EVENT_CLOSE_POP_UP(tab)
# define LOG_EVENT_CLOSE_POP_UP_IN(tab)
# define LOG_EVENT_BUTTON_CLICK(tab)
```

Obrázek 4.2: Makra pro logování otevírání, zavírání a klikání na tlačítka v rozbalovacích nabídkách

Logování logických událostí

Makra pro logování logických událostí lze vidět na Obrázku 4.3. První dvě makra slouží k zaznamenání propojení či rozpojení sekvencí a operátorů. Jako argument lze použít *CurveTab*, který byl pro spojení využit (bude zaznamenáno jméno sekvence / operátoru, ke které *CurveTab* patří). Přidání matice do sekvence je logováno pomocí události *LOG_EVENT_TAB_ADDED_AT_INDEX*. Přidání ostatních *Tabů* (například nový *Tab* do pracovní plochy, přesunutí ze sekvence do pracovní plochy atd.) je logováno pomocí makra *LOG_EVENT_TAB_ADDED*. Tato makra, stejně jako makro pro odebrání *Tabu*, potřebují jako argument *Tab*, který byl přidán, respektive odebrán, a *Tab*, do kterého bylo přidáno, respektive z něj odebráno. Poslední dvě makra slouží ke sledování objektů v sekvencích.

```
# define LOG_EVENT_CONNECT(startTab, endTab)
# define LOG_EVENT_DISCONNECT(startTab, endTab)
# define LOG_EVENT_TAB_ADDED_AT_INDEX(index,\
                                      addedTab,\
                                      addedToTab)
# define LOG_EVENT_TAB_ADDED(addedTab, addedToTab)
# define LOG_EVENT_TAB_REMOVED(removedTab, removedFromTab)
# define LOG_EVENT_OBJECT_ADDED(objectName, tab)
# define LOG_EVENT_OBJECT_REMOVED(tab)
```

Obrázek 4.3: Makra pro logování logických událostí v aplikaci

Logování polí v maticích

Dvě makra pro sledování polí v maticích (viz Obrázek 4.4) potřebují argument typu *Tab*, který identifikuje, o kterou matici se jedná. Dále index pole v matici, na kterém došlo ke změně a novou hodnotu změněného pole. První makro je používáno při změně pole pomocí klávesnice nebo rozbalovací nabídky, druhé makro při změně pole pomocí tažení myši.

```
# define LOG_EVENT_MATRIX_VALUE_UPDATE(matrix,\
                                       index,\
                                       newVal)
# define LOG_EVENT_MATRIX_VALUE_UPDATE_DRAG(matrix,\
                                            index,\
                                            newVal)
```

Obrázek 4.4: Makra pro logování změn hodnot v číselných polích matic

Ostatní makra

Další makra, která jsou součástí API logovacího systému jsou na Obrázku 4.5. Použití prvního makra bylo zmíněno v předchozí kapitole. Inicializační makro se nahraje definice zpráv ze souboru *events_definition.json* a nastaví časový formát pro logované zprávy. Poté zapíše zprávu o začátku logované seance do logu. Makro pro ukončení *Loggeru* zapíše do logu zprávu o konci logované seance. Makro *UPDATE_LOGGER* zpracovává vstup uživatele (kontroluje stisknutí tlačítek F1 - F5 pro zapínání a vypínání úrovní logování a logování zástupného záznamu o kroku tutoriálu).

```
# define ADD_TO_LOG_BUFFER(log_type, message, ...)
# define INIT_LOGGER(argc, argv)
# define END_LOGGER
# define UPDATE_LOGGER()
```

Obrázek 4.5: Ostatní makra, která jsou součástí API logovacího systému

4.3 Použití API v kódu

Po vytvoření logovacího systému je dalším krokem nalezení vhodných míst v kódu pro volání API logování definovaných událostí. Jednotlivé úrovně jsou logovány v různých částech programu.

Logování pozice kurzoru

Nejjednodušší je čisté logování pozice kurzoru. Celé toto logování probíhá ve třídě *InputController*, kde je při každém kliknutí zaznamenána pozice kurzoru a tlačítko, které bylo stisknuto, respektive uvolněno. Ve funkci update dochází k periodickému logování pozice kurzoru.

Logování rozbalovacích nabídek

Největším problémem bylo zvolit správné místo pro logování klikání v nabídkách a otevírání a zavírání rozbalovacích nabídek. Jelikož události pro kliknutí jsou definovány pro každé tlačítko zvlášť, logování přímo v nich by bylo prakticky neudržitelné z důvodu velkého množství různých událostí. Logování musí probíhat ve funkci *mouseUp*, která kliknutí zpracovává. Pokud je ve funcki *mouseUp* *Tab* vyhodnocen jako kliknutý, je voláno logovací API pro kliknutí tlačítka před případným voláním jeho události *onMouseUp*. My ovšem chceme logovat pouze kliknutí na tlačítka, ale třída *Button* používá děděnou funkci *mouseUp* základní třídy *Tab*. Aby nedocházelo ke zbytečnému kopírování kódu a přepisování zděděné funkce, která by byla obohacena pouze o logování, voláme logovací API pro každý kliknutý *Tab*. *Logger* poté vyhodnotí, zda se jedná o tlačítko a ostatní záznamy ignoruje.

Události otevření rozbalovacích nabídek (*PopUpMenu*) se zpracovávají ve třídě *TabSpace* (třída, která obsahuje všechny existující *Taby*), ve které jsou definovány všechny funkce pro zobrazení různých *PopUpMenu*. Každé zavolané funkci pro zobrazení *PopUpMenu* je předávána informace o *Tabu*, na který bylo kliknuto a tato informace je potřeba pro log. Události uzavření *PopUpMenu* se zaznamenávají ve funkci *mouseUp* třídy *PopUpMenu*. Při kliknutí mimo nabídku (při iterativním prohledávání vrstvy 0 - viz Kapitola 2.3) a při kliknutí na dále nerozbalitelnou položku v nabídce.

Logování logických událostí

Logování logických událostí je také poměrně přímočaré.

- Přidávání *Tabů* do pracovní plochy je logováno ve funkci *addTab* ve třídě *ScrollTab*, která reprezentuje pracovní plochu.
- Odebírání *Tabů* je logováno ve funkci *removeTab* v základní třídě *Tab*.
- Přidávání matic do sekvence je sledováno v metodě *addMatrix* třídy *TransformationForm* (tato třída reprezentuje sekvence), protože tato funkce má jako parametr pozici, na kterou je matice přidána.
- Sledování spojování uzlů v pracovní ploše je realizováno ve funkci *mouseUp* třídy *CurveTab*. Tato metoda kontroluje, zda se jedná o pokus o propojení uzlů a zda je pokus o spojení validní. Při kontrole validity je případně do paměti přidána zpráva o neúspěšnosti, která je následně zapsána s logem o samotném pokusu o zapojení.
- Rozpojení spojených uzlů je zaznamenáno v samotné logické funkci, která rozpojení realizuje (*CurveTab::unplug()*).
- Přidávání a odebírání 3D objektů ze scény je sledováno ve funkcích *addNewObjectToWorld* a *removeObjectFromWorld*.

Logování polí v maticích

Logování polí v maticích probíhá ve třídě *NumberBox*, která reprezentuje jednotlivá číselná pole v matici. V metodě *setVal*, která je volána při změně hodnoty v poli, je událost zaznamenána společně s informací o tom, které konkrétní pole konkrétní matice bylo změněno. Při změně hodnoty tažením myši dochází k průběžným změnám hodnot. To by zapříčinilo příliš velké množství záznamů v logu a proto logování neprobíhá, pokud je prováděna změna hodnoty pomocí tažení. V tomto případě je hodnota zaznamenána až při ukončení tažení ve funkci *mouseUp* třídy *NumberBox*.

4.4 Popis implementace vyhodnocovací aplikace

Po implementaci logovacího systému a začlenění jeho API do aplikace I3T můžeme začít pracovat na prototypu vyhodnocovací aplikace. Formát zpráv všech událostí zapisovaných do logu je definován v souboru *events_definition.json*, který se nachází v adresáři s aplikací I3T. Na základě těchto definic zpráv můžeme připravit vyhodnocovací aplikaci Log Viewer.

K vytvoření aplikace bude použit programovací jazyk Python. Pro jednodušší správu virtuálního prostředí zmíněného v Kapitole 3.2.2 byly vytvořeny dva skripty pro *PowerShell* [23]. Protože spouštěcí skript je napsaný v *PowerShellu*, může být použit pouze na systému Windows. To ovšem neznamená, že aplikaci nelze spustit na Linuxu. Znamená to pouze, že na Linuxu musíme zadat příkazy ručně, nebo vytvořit shellový skript se stejnou funkcionalitou.

Skript na Obrázku 4.6 slouží k úvodnímu nastavení virtuálního prostředí. Nejdříve pomocí správce balíků *pip*, který je součástí standardní instalace Pythonu, nainstaluje balík *venv* (v *pip* repozitáři pod jménem *virtualenv*). Následně vytvoří nové virtuální prostředí, aktivuje ho a uvnitř prostředí nainstaluje knihovny, specifikované v souboru *requirements.txt*. Posledním příkazem deaktivuje virtuální prostředí, aby příkazový řádek, ve kterém skript spouštíme, vrátil do původního stavu.

```
Set-Location $PSScriptRoot

python -m pip install --upgrade virtualenv
python -m venv env

& .\env\Scripts\Activate.ps1
python -m pip install --upgrade pip
python -m pip install -r requirements.txt
deactivate
```

Obrázek 4.6: *setup.ps1* - Skript k nastavení virtuálního prostředí

Skript na Obrázku 4.7 aktivuje virtuální prostředí, spustí program a po jeho skončení deaktivuje virtuální prostředí, aby vrátil příkazový řádek do původního stavu.

```
Set-Location $PSScriptRoot

& .\env\Scripts\Activate.ps1
python -m log_viewer
deactivate
```

Obrázek 4.7: *run.ps1* - Skript ke spuštění aplikace

Pozn.: Ke spuštění těchto skriptů je potřeba na některých systémech nejdříve povolit spouštění externích skriptů. Toho lze docílit jednoduše otevřením *PowerShellové* příkazové řádky s administrátorskými právy a zadáním příkazu „`Set-ExecutionPolicy RemoteSigned`".

Prvním krokem implementace aplikace je vytvoření datových tříd v balíku *model*. Dále bude potřeba převést záznam o uživatelské interakci do instancí těchto datových tříd. Následně můžeme začít pracovat na zobrazení dat v časových osách a připravit zbytek grafického rozhraní aplikace za pomoci knihovny *TkInter* [15].

Aby nedocházelo ke zmatku v názvosloví, slovem *záznam* bude v nadcházejících kapitolách myšlen celý záznam jedné sledované seance v I3T, tj. od spuštění aplikace do jejího vypnutí. Jeden soubor může obsazovat více *záznamů*. Naopak slovem *log* bude v následujících kapitolách myšlena jedna

„řádka" v *záznamu* (např. „Uživatel přidal matici X do sekvence Y", nebo „Uživatel klikl na tlačítko A").

4.4.1 Modelové třídy pro data

Modelové třídy byly vytvořeny podle návrhu diagramu tříd v Kapitole 3.3. Třídy značek *Marker*, respektive *MarkerLong* a *MarkerShort*, reprezentují jednotlivé události, které mají být zaneseny na časovou osu. Všechny značky obsahují časovou značku, reprezentující den a čas logu, zprávu z logu a barvu, kterou má být značka vykreslena. *MarkerLong* reprezentuje dlouhotrvající události jako například právě probíhající tutoriálový krok nebo probíhající logickou událost v aplikaci. Proto obsahuje dva atributy typu reálného čísla (*float*), které znázorňují začátek a konec značky. Třída *MarkerShort* obsahuje pouze jeden atribut typu *float*, protože reprezentuje jednorázovou událost, jejíž šířka bude v časové ose neměnná. Hodnotou těchto atributů je relativní vzdálenost od začátku záznamu v sekundách.

Tyto značky budou dále rozděleny do jednotlivých řádků (*Row*). Každý řádek má název, který bude vypsán v časové ose, a seznam značek, které do něho patří. Třída *EvalResult*, jejíž implementaci lze vidět na Obrázku 4.8, udává, jak bude vykreslená časová osa vypadat. Obsahuje informaci o délce záznamu v sekundách (*rel_finish*) a kvůli synchronizaci časových os také řádek s tutoriálovými kroky (*tutorials*). Informace o rozložení časové osy jsou uloženy v atributech *separate_rows* a *back_rows*. Atribut *separate_rows* je list řádků v pořadí, ve kterém mají být pod sebou vykresleny. Atribut *back_rows* je slovník, kde je klíčem index řádku a hodnotou řádek, který má být v řádku na tomto indexu vykreslen v pozadí. Pokud by mělo dojít ke změně rozložení řádků v časové ose, je potřeba změnit právě tyto atributy.

```
class EvalResult:
    def __init__(
        self,
        tutorials: Row,
        separate_rows: list,
        back_rows: dict,
        rel_finish: float,
        ):
        self.tutorials = tutorials
        self.rel_finish = rel_finish
        self.separate_rows = separate_rows
        self.back_rows = back_rows
```

Obrázek 4.8: Implementace datové třídy *EvalResult*

Třída *EvalResult* bude sloužit jako základní stavební kámen pro zobrazení časové osy. O převedení textového záznamu na instanci této třídy se postará třída *Evaluator*.

4.4.2 LogManager a Evaluator

Textový záznam uživatelské interakce je nutné převést do značek, ty rozdělit do správných řádků a z těchto řádků vytvořit instanci typu *EvalResult*, která může být zobrazena graficky. Pro tyto účely byl vytvořen balík *logic*. První třídou v balíku je třída *LogManger*. Tato třída se stará o nahrání logů ze souborů. Jako parametry konstruktoru očekává cestu k adresáři, kde má logy hledat, a zda má použít jako název záznamu název souboru, nebo časovou značku jeho prvního logu. Tyto dva argumenty jsou dodány z konfiguračního souboru. V jednom souboru se může nacházet více celistvých záznamů. V tomto případě je *LogManager* rozdělí na základě inicializační a konečné zprávy, jejíž formát zjistí ze souboru s definicemi logovaných zpráv, který se nachází v adresáři s aplikací I3T. Cesta k aplikaci je uložena v konfiguračním souboru. Jednotlivé logy jsou poté rozděleny do záznamů ve formě seznamu znakových řetězců, kde jeden string představuje jeden log. Každý tento list je poté přiřazen do slovníku pod názvem záznamu. Klíče tohoto slovníku (názvy záznamů) jsou poté zobrazeny v seznamu záznamů v menu.

Protože třída *Evaluator* z diagramu tříd z Kapitoly 3.3 neobsahuje žádné atributy a není žádný důvod, proč bychom ji potřebovali instancovat, byla implementována pouze jako funkce v samostatném souboru *evaluator.py*, kterou lze importovat odkudkoliv ze zbytku projektu. Tato funkce má za úkol transformovat záznam (seznam znakových řetězců), který bude poskytnut *LogManagerem* na instanci třídy *EvalResult*, která bude poté zobrazena jako časová osa.

Před vlastní transformací jsou nejdříve vytvořeny instance třídy *Row* pro pět řádků, které budeme zobrazovat (tutoriály, akce, kliknutí myši, zavření menu, otevřené menu). Každý log ze záznamu je převeden na instanci třídy *Marker* a přiřazen do patřičného řádku. Časové záznamy jednotlivých logů jsou pro tvorbu značek převedeny na relativní vzdálenost od začátku záznamu v sekundách a barva značky je nastavena podle konfiguračního souboru. Logy klikání na tlačítka a zavíraní menu vytváří instance třídy *MarkerShort*. Záznamy tutoriálů a logických událostí tvoří instance typu *MarkerLong*. Pokaždé, když se vyskytne log o logické události, považuje to *Evaluator* za ukončení této logické akce, na které uživatel až doposud pracoval. Konec předešlé akce tedy poslouží jako začátek nově nalezené akce. Logy tutoriálových kroků jsou v I3T zaznamenány při začátku práce na tutoriálovém kroku. Proto je při nalezení nového logu tutoriálu nastaven konec předešlého kroku tutoriálu na nově nalezený čas. Řádek otevřených menu je také plněn dlouhými značkami. Začátek značky je uložen, když je zpracován log o otevření menu, konec značky je uložen při zpracování logu o zavření menu.

4.4.3 Časová osa

Dalším krokem je vytvoření widgetu, který na vstupu dostane instanci třídy *EvalResult* a její řádky a značky zobrazí graficky. Za tímto účelem byla vytvořena třída *TimeLine*, která rozšiřuje třídu *Frame* z knihovny *TkInter*, aby ji bylo možné lehce začlenit do hierarchie ostatních widgetů v aplikaci.

Třída k vykreslení značek využívá plátno (*Canvas*), které je také součástí knihovny *TkInter*. Při vykreslování značek jsou vypočítávány souřadnice na plátně. Horizontální souřadnice jsou vypočítány podle časových záznamů ve značkách. Převod sekund na pixely je realizován pomocí proměnné *resolution*, která je specifikovaná v konfiguračním souboru, a faktoru přiblížení. Vertikální souřadnice jsou vypočítány podle pořadí řádku a výšek řádků, které jsou také specifikovány v konfiguračním souboru. Informace o barvě jsou obsažené ve značce samotné. Značka může mít také ohraničení, jehož šířka je také specifikována v konfiguračním souboru. S těmito informacemi můžeme využít funkci třídy *Canvas* pro vytvoření obdélníků na plátně (viz Obrázek 4.9). Dále je pro každou značku uložen její popis, který se zobrazí, pokud uživatel přejede kurzorem přes značku, když má stisknutou klávesu shift.

```
rectangle_id = self.__timeline.create_rectangle(
    coords,
    fill=color,
    width=border_width,
    outline=Colors.BLACK.value,
)
```

Obrázek 4.9: Využití funkce *create_rectangle* třídy Canvas

Kromě plátna se značkami obsahuje třída ještě druhé plátno, na kterém je v určitých intervalech znázorněn uplynulý čas od začátku záznamu. Pro toto byla využita implementace časové osy z knihovny *ttkwidgets* [20].

Pro funkcionalitu přibližování byla využita možnost přiřazení funkcí, které se volají při specifikovaných událostech, což je implementováno pro všechny widgety z knihovny *TkInter*. S plátnem časové osy byla provázána funkce pro změnu faktoru přiblížení (viz Obrázek 4.10) pomocí `self.__timeline.bind("<MouseWheel>", self.__on_zoom)`. Tato funkce se volá při točení kolečkem myši nad časovou osou.

První funkce `__on_zoom(self, event: tk.Event)` zpracovává událost otočení kolečkem myši (tk.Event). Pomocí pozice kurzoru myši vůči plátnu zjistí, zda se kurzor nachází v levé, prostřední nebo pravé části plátna (kvůli přiblížení tímto směrem) a poté volá funkci `__zoom(self, event: tk.Event, direction: str)`. Tato funkce upraví faktor přiblížení funkcemi `__zoom_in()` a `__zoom_out()`, které kontrolují maximální a minimální hodnoty přiblíení. Poté zavolá překreslení časové osy s novou hodnotou přiblížení a posune scrollbar směrem ke kurzoru pomocí funkce `__scrollbar_on_zoom(self, scrollbar_pos, direction)`.

Synchronizace časových os probíhá pomocí provolávání funkcí třídy *MainView*. Při každém pohybu po časové ose (pomocí shift + kolečko myši nebo scrollbaru) dochází k volání funkce pro kontrolu aktuálního kroku tutoriálu, kterou lze vidět na Obrázku 4.11. Tato funkce kontroluje, zda jsme se posunuli na jiný tutoriálový krok, a případně volá funkci *___notify_synced*. Funkce *___notify_synced* je ukazatel na funkci třídy *MainView*, která se pokusí zarov-

nat ostatní synchronizované časové osy voláním jejich synchronizační funkce. Synchronizační funkci lze vidět na Obrázku 4.12.

```
def __on_zoom(self, event: tk.Event) -> None:
    left_threshold, right_threshold = 0.4, 0.6
    relative_x = (
        self.__canvas_timeline_scroll.winfo_pointerx()
        - self.__canvas_timeline_scroll.winfo_rootx()
    )
    fraction = relative_x / self._get_tl_window_width()
    direction = (
        tk.LEFT
        if fraction < left_threshold
        else (
            tk.CENTER
            if fraction < right_threshold
            else tk.RIGHT
        )
    )
    self.__zoom(event, direction)

def __zoom(self, event: tk.Event, direction: str) -> None:
    scrollbar_pos = self.__scrollbar.get()
    last_zoom = self._zoom
    if event.delta < 0:
        self.__zoom_out()
    elif event.delta > 0:
        self.__zoom_in()
    if last_zoom != self.__zoom_factor:
        self.__draw_timeline()
        self.__scrollbar_on_zoom(scrollbar_pos, direction)
```

Obrázek 4.10: Funkce pro změnu faktoru přiblížení *__on_zoom* a *__zoom*

```
def __notify_tutorial_step_change(self) -> None:
    current_step = self.__at_step()
    if current_step != (
        self.__last_tutorial_step and current_step != None
    ):
        self.__notify_synced(
            self, current_step.message, self.__zoom_factor
        )
        self.__last_tutorial_step = current_step
```

Obrázek 4.11: Funkce pro kontrolu změny tutoriálového kroku

```
def align_to_tutorial(
    self, tutorial_name: str, new_zoom: float
) -> None:
    try:
        index = [
            marker.message
            for marker in self.__actions.tutorials.markers
        ].index(tutorial_name)
        tutorial_start = self.__actions.tutorials.markers[
            index
        ].start
        if new_zoom != self._zoom:
            self.__zoom = new_zoom
            self.__draw_timeline()
        self.__scroll_to_second(tutorial_start)
    except ValueError:
        print("Tutorial step not in this timeline")
```

Obrázek 4.12: Funkce pro synchronizaci časové osy *align_to_tutorial*

4.4.4 TkInter GUI

Pomocí *Evaluatoru* můžeme vytvářet instance modelových tříd z textového záznamu interakce a máme i možnost tuto interakci graficky zobrazit pomocí časové osy. Zbývá nám pouze složit tyto prvky dohromady a vytvořit zbytek grafického rozhraní aplikace Log Viewer.

Hlavní třídou, která zajišťuje grafické rozhraní, je třída *MainView*. V této třídě jsou vytvořeny a spravovány všechny widgety v aplikaci. Třída se také stará o interakci s uživatelem (otevírání a zavírání časových os a jejich synchronizaci) a drží instance třídy *LogManager* pro seznam nalezených záznamů a scénářů. V konstruktoru třídy jsou vytvořeny všechny widgety na obrazovce a s některými je provázána funkce (seznam záznamů a scénářů - funkce pro vytvoření zvolené časové osy).

Pro widgety okno (*Frame*), seznam záznamů (*Treeview*) a drop down menu scénářů (*Combobox*) byly v souboru *ui_objects.py* vytvořeny obalové třídy, které tyto widgety mírně přizpůsobí pro naše účely (přidání scrollbaru do *Treeview* a jejich automatické zobrazení). V souboru *ui_objects.py* byla také vytvořena třída *MyTimeLine*. Tato třída obaluje třídu časové osy z Kapitoly 4.4.3. Přidá k ní nadpis a tlačítka pro zavření a sychronizaci. S těmito tlačítky prováže ovládací funkce třídy *MainView*.

Na Obrázku 4.13 lze vidět výsledné grafické rozhraní aplikace. Narozdíl od návrhu wireframu z Obrázku 3.3 obsahuje ve třetí časové ose zvýraznění právě otevřené rozbalovací nabídky z toho důvodu, že požadavek na zobrazení otevřených menu byl přidán dodatečně během implementace.

Obrázek 4.13: Snímek obrazovky aplikace Log Viewer

4.4.5 Konfigurační soubor

Konfigurační soubor, který byl několikrát zmíněn v předcházejících kapitolách obsahuje proměnné, které by mohl uživatel chtít měnit, aby přizpůsobil aplikaci svým potřebám. Mezi nejdůležitější konfigurovatelné proměnné patří:

- výšky jednotlivých řádků v časové ose
- statická šířka značek kliknutí
- šířka ohraničení jednotlivých značek
- barvy jednotlivých druhů značek
- změna přiblížení pro jedno otočení kolečka myši
- formát zprávy značek

Nejdůležitější položka konfigurace je však cesta k I3T, která musí být specifikována kvůli získání definic logovaných zpráv ze souboru *events_definition.json*, které jsou použity při vyhodnocení záznamů.

Kapitola 5
Testování

V kapitole bude navržen postup sběru a vyhodnocení záznamů uživatelské interakce za účelem vývoje rozhraní i přípravy tutoriálových úloh. Dále bude logovací systém a vyhodnocovací aplikace Log Viewer otestována na třech vzorových úlohách a na základě tohoto testování bude zhodnocena užitečnost aplikace Log Viewer.

5.1 Testovací strategie pro vývoj rozhraní

Při sběru dat za účelem vývoje rozhraní je důležité, aby testované osoby neznaly aplikaci I3T. Rozhraní aplikace by mělo být intuitivní a uživatelé by měli být schopni se v něm zorientovat i bez předcházejícího seznámení s aplikací.

Úkoly v aplikaci I3T musejí být dostatečně jednoduché. To proto, aby případná neznalost teorie transformací testovaných osob nezkreslovala výsledky testů. Úkoly by měly dostatečně detailně popisovat, co je potřeba udělat (ale ne do míry jednotlivých kliknutí). Správný krok v úkolu by mohl vypadat takto:

- Posuňte krychli o 1 nahoru přidáním matice translate s hodnotou 1 v poli v druhé řádce a čtvrtém sloupci.

Takový úkol popisuje, co je cílem a jak tohoto cíle dosáhnout. I kdyby uživatel nevěděl, že k posunutí objektů slouží translační matice a k posunutí směrem nahoru dojde posunem po ose Y, je z popisu úkolu schopen úkol splnit. Jedinou jeho překážkou je nalezení správných nástrojů (přidání matice) v aplikaci I3T.

Nesprávný krok v úkolu by vypadal například takto:

- Posuňte krychli o 1 nahoru.
- Posuňte krychli o 1 nahoru přidáním matice translate (kliknutí pravým tlačítkem myši na pracovní plochu > „transformation” > „translation”) a změňte hodnotu pole v druhé řádce a čtvrtém sloupci na 1

První verze tohoto kroku neobsahuje dostatečné množství informací. V případě, že uživatel nebude mít dostatečné teoretické základy, nebude vědět, jak

posunout krychli. Z takového záznamu nepoznáme, zda je problém v grafickém rozhraní aplikace nebo v neznalosti transformací na straně uživatele. Druhý příklad špatného kroku má opačný problém. Pro uživatele je navádějící až příliš. Uživatel prakticky nemá šanci se sám zorientovat v grafickém rozhraní I3T.

Po získání záznamů následuje jejich vyhodnocení. Výsledkem tohoto vyhodnocení bude nalezení částí rozhraní, které jsou problematické. Problematické části lze identifikovat pomocí „špatných" kliknutí v menu (např. uživatel měl přidat translační matici, ale klikl v menu na nabídku operátorů). K tomuto vyhodnocení může být použita aplikace Log Viewer. Při vyhodnocení lze s výhodou využít zejména možnost zobrazení zlatého průchodu scénářem paralelně se záznamem uživatele. Při porovnání těchto dvou záznamů bude na první pohled vidět, kde kliknutí přebývají.

Dalším znakem špatného rozhraní může být dlouhá časová prodleva při plnění úkolu. V tomto případě lze zjistit, kde uživatel hledal v I3T nástroje ke splnění úkolu pomocí pozice kurzoru, kterou logovací systém periodicky zaznamenává. Log Viewer však nedokáže tuto informaci zpracovat, takže pozice kurzoru v čase plnění úkolu bude muset být dohledána ručně.

5.2 Testovací strategie pro vyhodnocení tutoriálových úloh

Při tvorbě tutoriálových úloh je zásadní úroveň teoretických znalostí cílové skupiny budoucích uživatelů a podle těchto znalostí se odvíjí, do jaké míry detailu jsou v tutoriálu popsány jednotlivé kroky úloh. Cílem tohoto testování je přizpůsobit tuto míru detailu cílové skupině uživatelů nebo se zaměřit na zlepšení jejich konkrétních teoretických znalostí při výuce.

V kapitole zabývající se požadavky na logování 2.2 bylo zjištěno, že při sledování řešení úloh je důležité se zaměřit na logické události v aplikaci. Narozdíl od hledání „špatných" kliknutí v případě testování za účelem vývoje rozhraní (viz Kapitola 5.1), při testování tutoriálových úloh je potřeba nalézt „špatné" logické události v aplikaci (např. uživatel má za úkol přidat translační matici, ale nejdříve přidal matici scale, kterou poté smazal), nebo časové prodlevy při plnění úkolu. Tyto časové prodlevy mohou být způsobeny tím, že uživatel neví, co má dělat, a proto tápe, ale také špatným rozhraním aplikace (uživatel ví, že má přidat matici translate, ale neví, kde ji najít). Tyto případy lze jednoduše filtrovat, pokud bude kromě logických událostí logováno také klikání na tlačítka v menu. V aplikaci Log Viewer poté při porovnání uživatele se vzorovým scénářem bude zřejmé, že při logické události přebývají zbytečná kliknutí a uživatel pravděpodobně bloudil v nabídce.

5.3 Testování na konkrétních úlohách

Tutoriálové úlohy zatím nejsou součástí aktuální verze aplikace I3T. Proto jsou v aplikaci simulovány stiskem klávesy F5, která zapíše do logovacího souboru

zástupný záznam s číslem úkolu. Při plnění úlohy podle vytištěného tutoriálu I3T je proto nutné při začátku práce na novém kroku stisknout klávesu F5 pro zaznamenání postupu v úloze. Aby mohla být využita funkcionalita zobrazení a synchronizace tutoriálových kroků v Log Vieweru, byl vytvořen skript, který zástupné záznamy o plnění tutoriálu v souboru s logem nahradí konkrétními kroky sledované lekce specifikovanými v jiném souboru. Zároveň také posune časovou značku tutoriálového kroku na časovou značku předcházejícího záznamu, aby začátky tutoriálových logů souhlasily s událostmi v aplikaci (tzn. aby tutoriálový krok nezačal uprostřed logické události).

Testování konkrétních úloh bylo provedeno na dvou studentech ČVUT FEL, kteří minulý rok absolvovali předmět Programování grafiky. Díky tomu by měli mít dostatečné teoretické znalosti v oblasti transformací. S aplikací I3T se také setkali, takže znali základy jejího ovládání, ale už téměř rok ji neviděli. Výukové lekce (viz Příloha D) byly dodány vedoucím, vzorové scénáře (ideální sekvence kroků ke splnění lekce) byly vytvořeny autorem práce.

Testování proběhlo za účelem ověření funkcionality logovacího systému a aplikace Log Viewer a jako ukázka toho, jak lze s aplikací pracovat. Účelem testování není nalezení chyb v grafickém rozhraní I3T nebo optimalizace tutoriálových úloh, protože to není tématem práce.

Pozn.: Pokud jsou výřezy jednotlivých událostí odděleny čarou a vlnovkou, znamená to, že se jedná o po sobě jdoucí události jedné časové osy. Pokud výřezy nejsou takto odděleny, jedná se o výřezy z různých časových os.

5.3.1 Tutoriálová úloha č. 2 - Základy transformací

V druhé úloze v tutoriálu k I3T (viz Příloha D) se uživatel naučí aplikovat transformace na objekty a dozví se, jak transformace reprezentovat pomocí matic. V prvním kroku této lekce byla v tutoriálu objevena chyba. Při testování na ni byl testovaný uživatel upozorněn. Z úlohy byly vytaženy jednotlivé kroky, které byly následně pomocí skriptu doplněny do souboru s logy z plnění tutoriálové úlohy a do vzorového scénáře úlohy. Tyto kroky jsou následující (poznámky kurzivou u byly doplněny pro lepší srozumitelnost bez znalosti kontextu celého textu tutoriálu):

1. Klikněte pravým tlačítkem myši na workspace, ve vyskakovacím menu zvolte sequence

2. Klikněte pravým tlačítkem myši na sekvenci, ve vyskakovacím menu zvolte bind object > basics > cube > white *Druhá lekce navazuje na první lekci, při které byla přidána bílá krychle*

3. Klikněte pravým tlačítkem myši na workspace, ve vyskakovacím menu zvolte sequence *Tutoriál instruuje přidání krychle přímo z pracovní plochy, což není možné*

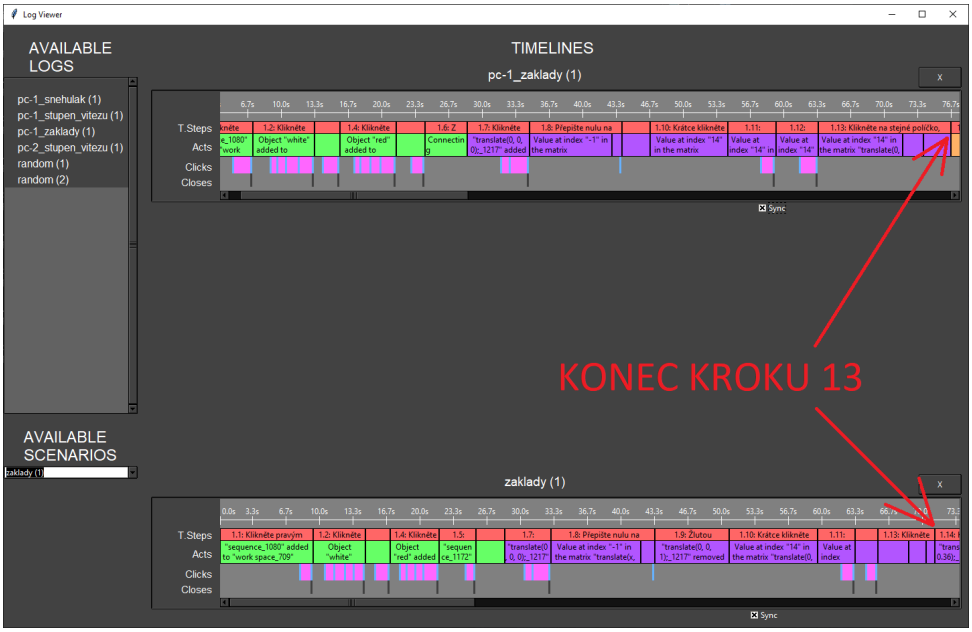
4. Klikněte pravým tlačítkem myši na sekvenci, ve vyskakovacím menu zvolte bind object > basics > cube > red *Uživatel nevidí červenou krychli, protože je na stejném místě jako bílá krychle*

5. Klikněte pravým tlačítkem myši na workspace, ve vyskakovacím menu zvolte sequence
6. Z pravého křížku na sekvenci vytáhněte myší bílý drát a zapojte jej do křížku vlevo na krabičce s červenou krychlí
7. Klikněte pravým tlačítkem myši na workspace, ve vyskakovacím menu zvolte transformation > translation
8. Přepište nulu na jedničku v poli „z:" a klikněte na ok
9. Žlutou krabičku s nadpisem translate(0,0,1) přetáhněte do připravené zapojené sekvence
10. Krátce klikněte myší (L-click) na třetí políčko v posledním sloupci matice ve žluté krabičce a změňte hodnotu z 1 na 2 pomocí klávesnice
11. Klikněte na stejné políčko pravým tlačítkem myši a ve vyskakovacím okně set... vyberte 1
12. Zkuste vybrat -1
13. Klikněte na stejné políčko, držte jej stlačené a pohybujte myší, změňte takto i ostatní zelené hodnoty
14. Klikněte pravým tlačítkem na workspace, ve vyskakovacím menu transformation vyzkoušejte postupně matice uniform scale, scale, eulerAngle, rotate, potvrďte počáteční hodnoty a vložte matici do sekvence, předchozí matici vyndejte tažením levým tlačítkem myši za horní lištu matice
15. Smažte všechny sekvence a matice z workspace
16. Vložte tři krychle (modrou, bílou, tyrkysovou)
17. Poskládejte je na sebe podél osy Y

Pro úlohu byl vytvořen vzorový scénář a poté záznam průchodu testovaného uživatele. Tyto dva záznamy byly následně zobrazeny v aplikaci Log Viewer.

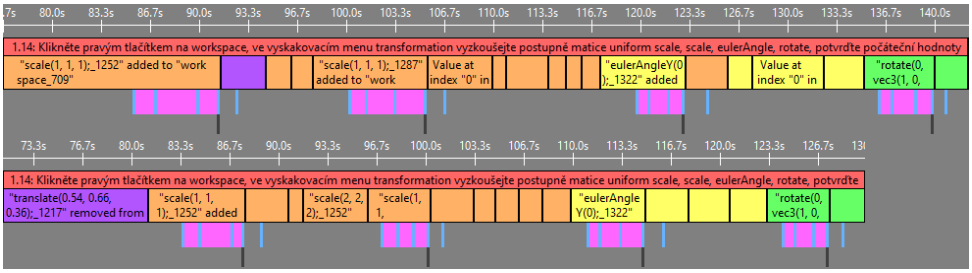
Na Obrázku 5.1 vidíme první krok vyhodnocení záznamů. Nejdříve oddálíme záznam testovaného uživatele (osa nahoře) tak, abychom viděli prvních třináct červených bloků v prvním řádku časové osy, tzn. prvních třináct kroků lekce (kroky vedou uživatele klik po kliknutí k aplikaci transformace). Pomocí synchronizace jsme automaticky nastavili míru přiblížení na stejně velký časový krok i na vzorovém záznamu (časová osa dole). Vidíme, že prvních třináct kroků pracoval uživatel téměř stejně rychle jako vzorový scénář (byl pouze o cca 5 sekund pomalejší).

Synchronizované osy posuneme na čtrnáctý krok (vyzkoušení dalších transformačních matic). Na výřezu čtrnáctého kroku na Obrázku 5.2 vidíme, že uživatel (časová osa nahoře) byl opět o pár sekund pomalejší (červený blok s krokem úlohy v horní časové ose je delší než v ose dole). Když se zaměříme na druhý řádek v časových osách, všimneme si, že uživatel prováděl akce



Obrázek 5.1: Úloha č. 2 zobrazena v aplikaci Log Viewer

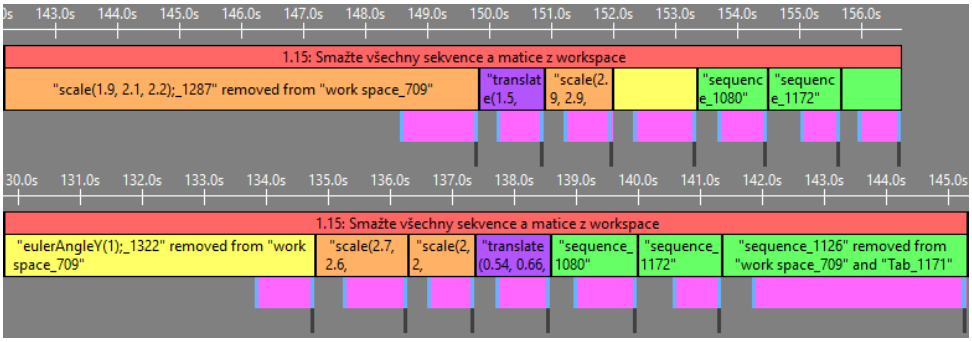
v jiném pořadí než vzorový scénář. To můžeme vidět na první pohled díky barevnému rozlišení akcí týkajících se různých matic. To je pravděpodobně způsobeno tím, že čtrnáctý krok úlohy již nenavádí uživatele klik po kliknutí, ale dává mu více volnosti.



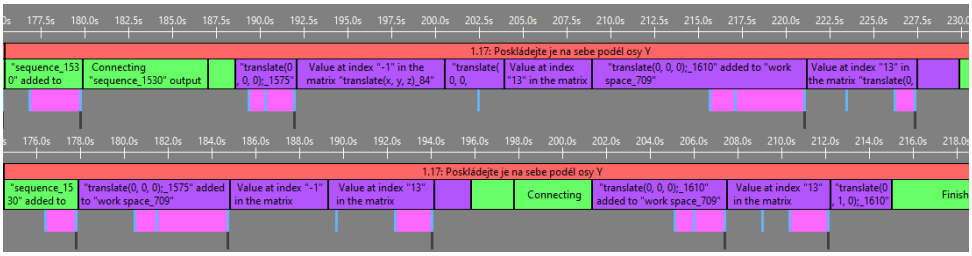
Obrázek 5.2: Výřez časové osy uživatele a vzorového scénáře (úloha č. 2, krok 14)

Dále posuneme synchronizované osy na patnáctý krok (vymazaní všech krychlí a matic). Na výřezu patnáctého kroku na Obrázku 5.3 vidíme, že uživatel byl dokonce rychlejší než vzorový scénář. Stejně jako v předchozím kroku si můžeme všimnout, že matice vymazal v odlišném pořadí (podle barevných akcí v druhém řádku).

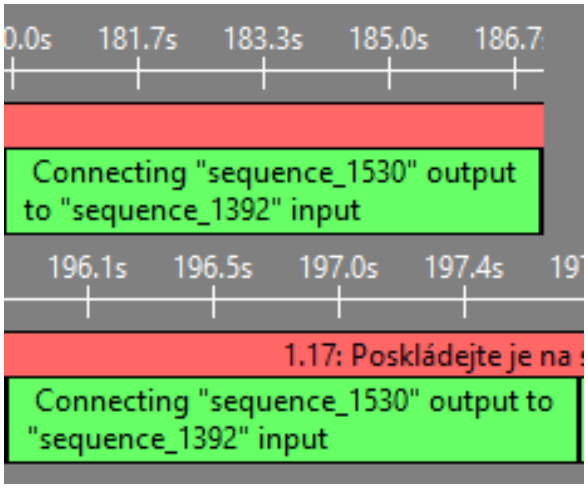
Stejné chování pozorujeme při posledním kroku úlohy (poskládání krychlí na sebe). Na Obrázku 5.4 vidíme, že barevné bloky v druhém řádku obou časových os mají různé pořadí. Při bližším prozkoumání akcí (viz Obrázek 5.5) vidíme, že se jedná o akci propojení sekvencí. Ve vzorovém záznamu byla translační matice nejdříve přidána do sekvence, až poté došlo k propojení sekvencí, uživatel vykonal tyto akce v opačném pořadí.



Obrázek 5.3: Výřez časové osy uživatele a vzorového scénáře (úloha č. 2, krok 15)



Obrázek 5.4: Výřez časové osy uživatele a vzorového scénáře (úloha č. 2, krok 17)



Obrázek 5.5: Výřez událostí z Obrázku 5.4 provedených v různém pořadí (začátek horní časové osy, prostředek dolní časové osy)

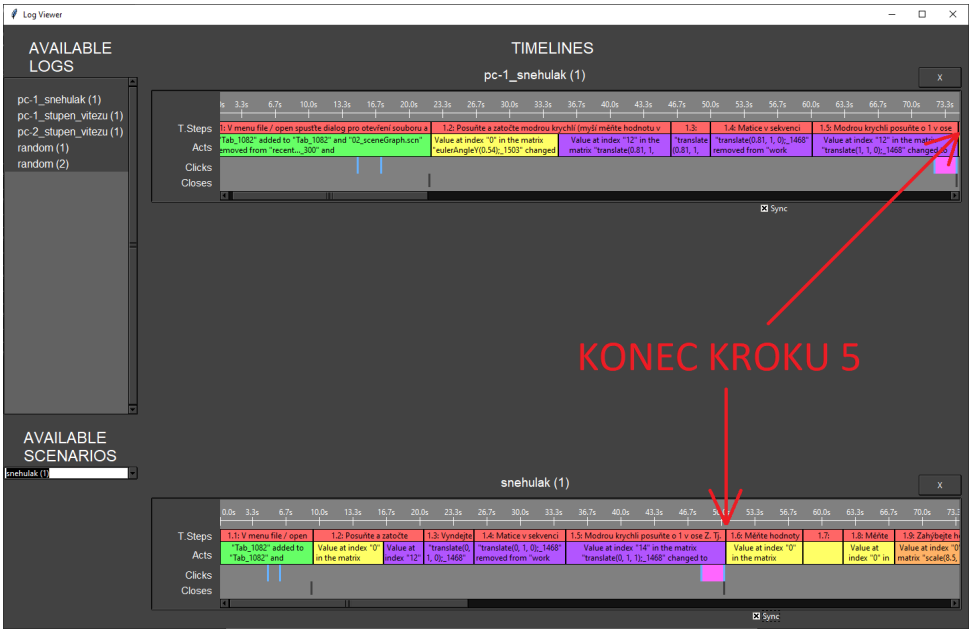
5.3.2 Tutoriálová úloha č. 3 - Základy skládání transformací

Třetí úloha v tutoriálu k I3T (viz Příloha D) seznámí uživatele s tím, jak transformace skládat za sebou do sekvence a jak propojovat sekvence do grafu scény. Z matematického pohledu jde o násobení matic ve správném pořadí. Při plnění úlohy měli uživatelé k dispozici celý tutoriál, aby se mohli podívat do první lekce na ovládání, pokud jej už zapomněli. Z úlohy byly vytaženy jednotlivé kroky, které byly následně pomocí skriptu doplněny do souboru s logy z plnění tutoriálové úlohy a do vzorového scénáře úlohy. Tyto kroky jsou následující (poznámky kurzivou u byly doplněny pro lepší srozumitelnost bez znalosti kontextu celého textu tutoriálu):

1. V menu file / open spusťte dialog pro otevření souboru a v adresáři <adresář s programem I3T> / data / scenes vyberte 02_sceneGraph.scn (*Scéna obsahuje tři krychle s různými transformacemi a propojeními*)
2. Posuňte a zatočte modrou krychlí (myší měňte hodnotu v některém ze zelených políček matice eulerAngleY, myší měňte hodnoty v matici translate) (*Sekvence s modrou krychlí obsahuje rotační a translační matici*)
3. Vyndejte jednu z matic ze sekvence
4. Matice v sekvenci dostaňte do stavu TR (*pořadí matic Translation - Rotation*)
5. Modrou krychli posuňte o 1 v ose Z. Tj. do bodu (0, 1, 1)
6. Měňte hodnoty v matici otáčení (*Uživatel vidí, že se krychle otáčí okolo své osy, protože se rotace aplikuje jako první*)
7. Změňte pořadí matic
8. Měňte hodnoty v matici otáčení (*Uživatel vidí, že se krychle otáčí okolo středu, protože se rotace aplikuje po translaci*)
9. Zahýbejte hodnotami matice scale u bílé krychle
10. Zahýbejte hodnotami matice translate červené krychle (*Uživatel vidí, že se pohybují i další propojené krychle*)
11. Zahýbejte hodnotami matice translate úplně vlevo (*Uživatel vidí, se hýbají všechny objekty*)
12. Vložte do scény zelenou krychli
13. Nastavte velikosti krychlí pomocí matice uniform scale na červená - 1, zelená $\sqrt{2}/2$, modrá 0.5
14. Poskládejte je na sebe podél osy Y
15. Sněhuláka postavte na znak plus na bílé podstavě

Pro úlohu byl vytvořen vzorový scénář a poté záznam průchodu testovaného uživatele. Tyto dva záznamy byly následně zobrazeny v aplikaci Log Viewer.

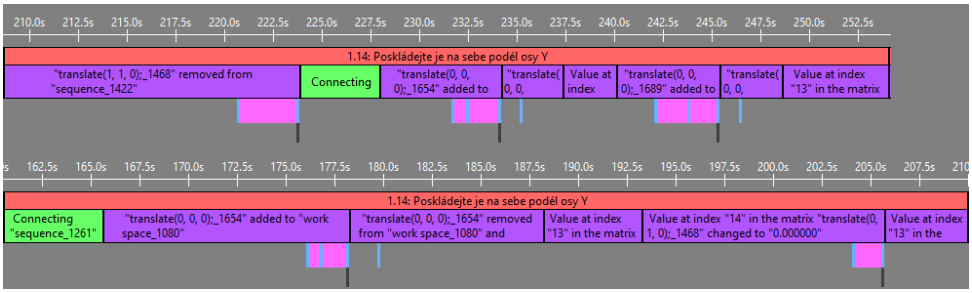
Na Obrázku 5.6 je vidět první krok vyhodnocení. Uživatelův záznam (časová osa nahoře) jsme oddálili tak, abychom viděli prvních pět kroků tutoriálu (červené bloky v prvním řádku). Pomocí synchronizace jsme automaticky nastavili míru přiblížení na stejně velký časový krok i na vzorovém záznamu (časová osa dole). Nyní na první pohled vidíme, že prvních pět červených bloků zabírá u uživatele zhruba o třetinu více času než ve vzorovém scénáři. Porovnáme tedy další řádky (zaznamenané logické události v aplikaci, kliknutí na tlačítka, zavření nabídek) a pokusíme se zjistit, čím je to způsobeno. Při porovnání dalších řádek zjistíme, že události i kliknutí jsou stejné. Uživatel pracoval pomaleji, ale nenarazil na žádné problémy.



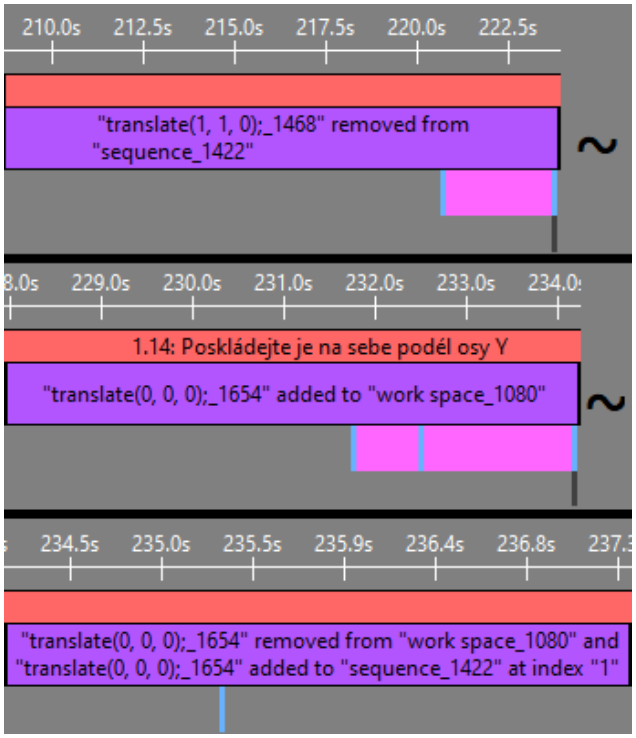
Obrázek 5.6: Úloha č. 3 zobrazena v aplikaci Log Viewer

Další úkoly proběhly stejným způsobem. Za povšimnutí stojí poslední dva úkoly. Na Obrázku 5.7 vidíme výřez úkolu poskládání krychlí na sebe podle osy Y. Uživatel (časová osa nahoře) byl sice o něco rychlejší než vzorový záznam (osa dole), ale místo pěti akcí týkajících se translačních matic (fialové bloky) jich vykonal sedm.

Při detailnějším prozkoumání zjistíme, že uživatel nejdříve vymazal ze sekvence s ID 1422 translační matici. Poté si pravděpodobně uvědomil, že ji tam potřebuje, tak přidal novou a znovu ji do této sekvence vložil (viz tři výřezy jednotlivých událostí na Obrázku 5.8). Pozn.: Pokud se v jednom bloku nachází zároveň dvojice zpráv - jedna o odstranění a druhá o přidání matice (viz poslední blok na Obrázku 5.8), znamená to, že byla matice přesunuta.

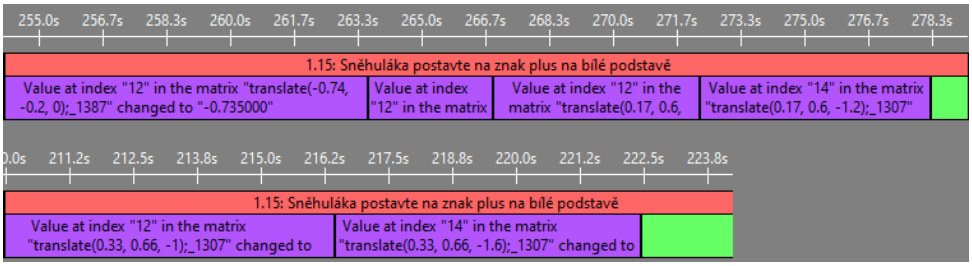


Obrázek 5.7: Výřez časové osy uživatele a vzorového scénáře (úloha č. 3, krok 14)

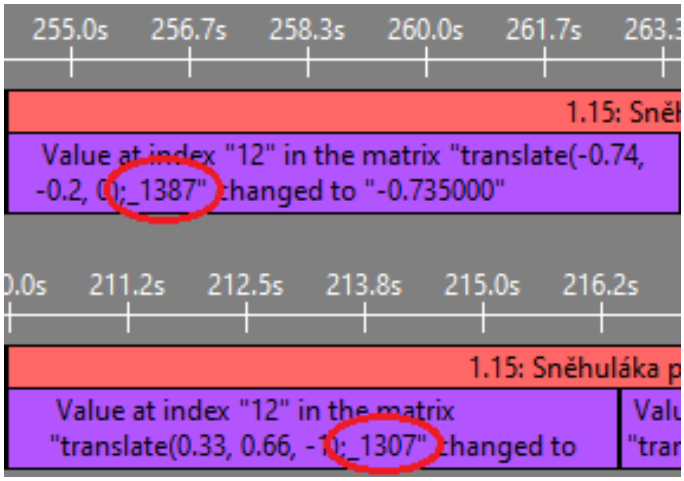


Obrázek 5.8: Tři výřezy po sobě jdoucích událostí ze záznamu uživatele z úlohy č. 3, kroku 14

Při plnění posledního úkolu (posunutí celého sněhuláka) na Obrázku 5.9 vidíme opět u uživatele (časová osa nahoře) větší množství akcí týchajících se translačních matic. Ve scénáři lze vidět, že matice, která pohybuje celým sněhulákem, má ID 1307 (časová osa dole na Obrázku 5.10). Uživatel měnil hodnoty ve správném poli matice (index "12"), ale nejdříve v matici s ID 1387 (časová osa nahoře na Obrázku 5.10). ID matice je jediný způsob, kterým logovací systém matice rozlišuje. Z jednoho logu se tedy nedozvíme o kterou matici se jedná (se kterou krychlí tato matice pohybuje).



Obrázek 5.9: Výřez časové osy uživatele a vzorového scénáře (úloha č. 3, krok 15)



Obrázek 5.10: Dva výřezy událostí ze záznamu uživatele a ze záznamu vzorového scénáře z úlohy č. 3, kroku 15

5.3.3 Vlastní úloha - Stupeň vítězů

Úloha spočívá ve vytvoření jednoduchého stupně vítězů pomocí posunu a změny velikosti tří krychlí. Úloha ověřuje základní znalosti studenta v oblasti transformací. Kroky úlohy jsou následující:

1. Otevřete prázdnou scénu
2. Přidejte sekvenci do workspace
3. Do sekvence přidejte modrou krychli
4. Modrou krychli posuňte o 0.5 nahoru
5. Přidejte druhou sekvenci do workspace
6. Do sekvence přidejte červenou krychli
7. Změňte výšku červené krychle na 0.7
8. Červenou krychli posuňte o 0.5 nahoru a o -1 po ose X (dejte pozor na pořadí transformačních matic)
9. Přidejte třetí sekvenci do workspace

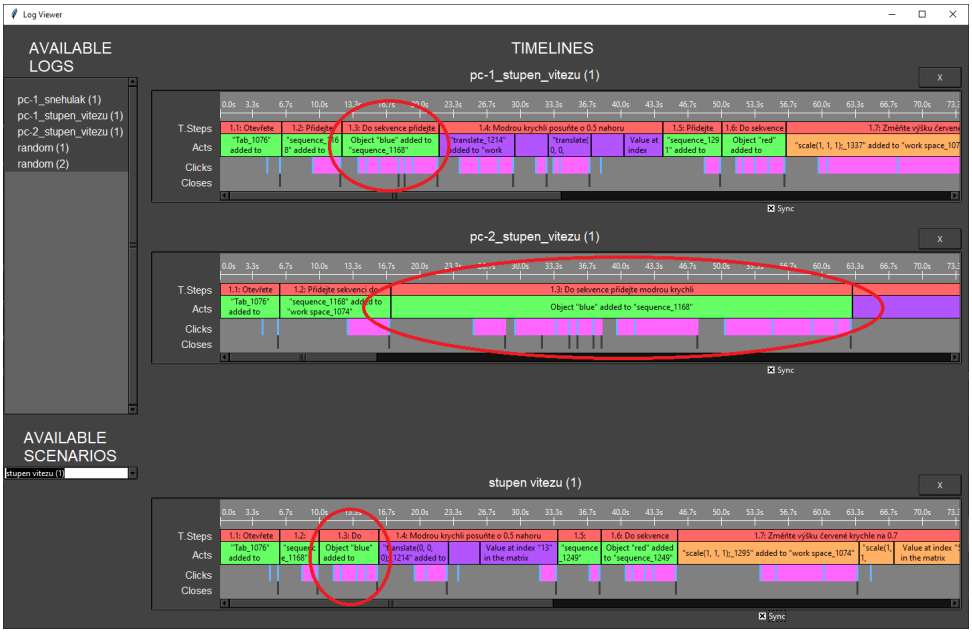
10. Do sekvence přidejte zelenou krychli

11. Změňte výšku zelené krychle na 0.5

12. Zelenou krychli posuňte o 0.5 nahoru a o 1 po ose X (dejte pozor na pořadí transformačních matic)

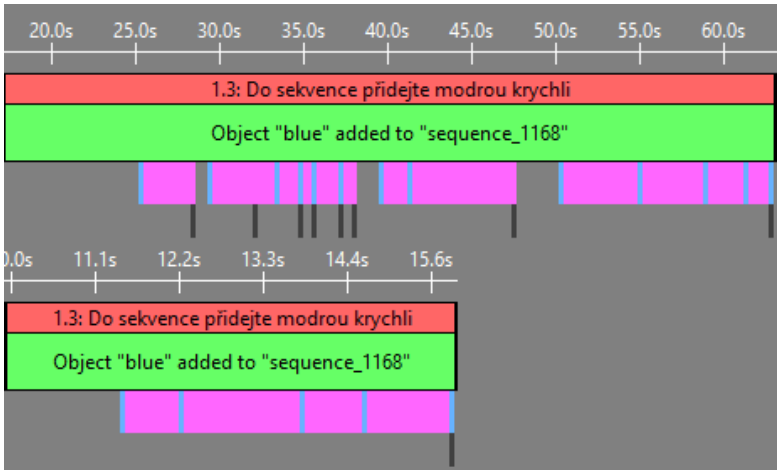
Pro tutoriál byl vytvořen vzorový scénář, poté byl zaznamenán průchod uživatele, který testoval předcházející úlohy, a průchod druhého uživatele, který první dvě úlohy neabsolvoval. Tyto dva záznamy společně se vzorovým scénářem byly následně zobrazeny v aplikaci Log Viewer.

Na začátku vyhodnocení jsme oddálili prostřední časovou osu (druhý uživatel) tak, abychom viděli celé první tři kroky, a ostatní osy synchronizovali podle ní. Pokud se zaměříme na první řádek časových os, který znázorňuje trvání kroků tutoriálu, vidíme, že druhý uživatel (prostřední osa) měl problém se třetím úkolem (přidání modré krychle). Na Obrázku 5.11 vidíme, že třetí krok zabírá v záznamu druhého uživatele několikanásobně více času, než v záznamu prvního uživatele a ve vzorovém scénáři.



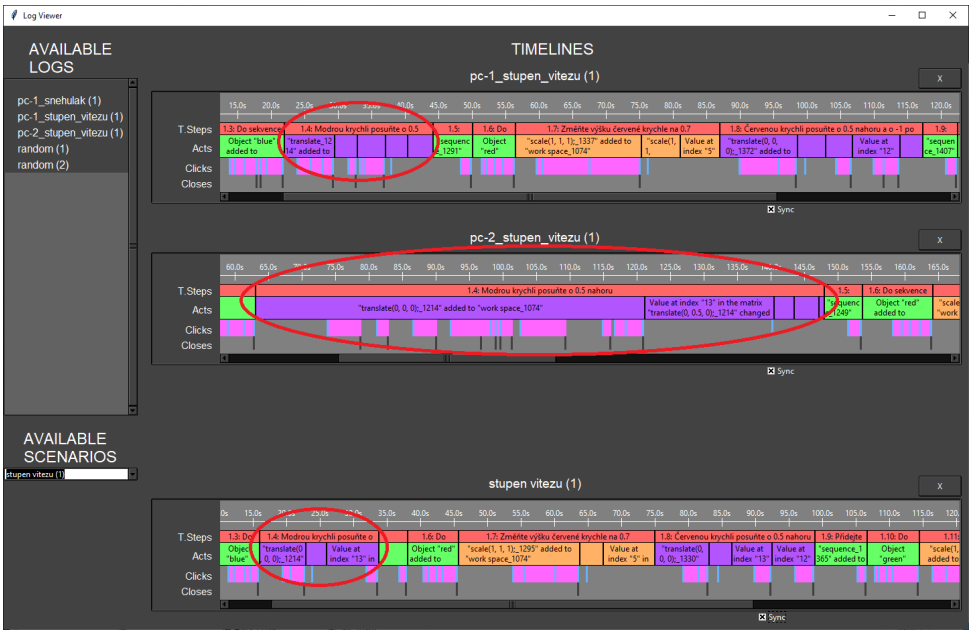
Obrázek 5.11: Vlastní úloha zobrazena v aplikaci Log Viewer

Deaktivujeme synchronizaci a časové osy přiblížíme na třetí tutoriálový krok, abychom je mohli lépe prozkoumat. Zaměříme se na třetí a čtvrtý řádek, které obsahují jednotlivá kliknutí a zavření nabídek. Na Obrázku 5.12 vidíme, že ve vzorovém scénáři (časová osa dole) bylo provedeno celkem pět kliknutí na tlačítka v jedné rozbalovací nabídce, zatímco druhý uživatel (časová osa nahoře) klikl celkem třináctkrát ve čtyřech nabídkách, než se mu podařilo krychli přidat.



Obrázek 5.12: Výřez třetího kroku vlastní úlohy ze záznamu druhého uživatele a vzorového scénáře

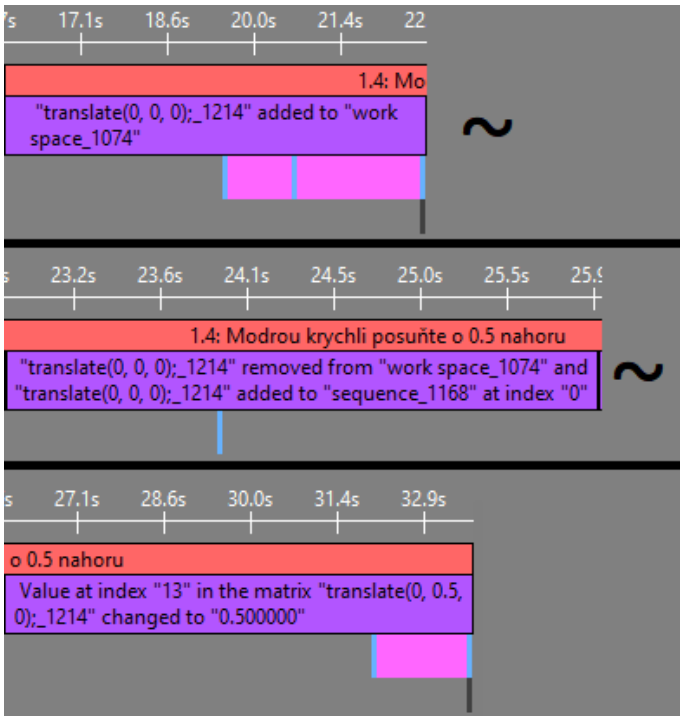
Časové osy synchronizujeme a posuneme dále. Naprosto stejný problém můžeme pozorovat při plnění čtvrého úkolu (posunutí krychle). Na Obrázku 5.13 lze vidět, že trvání čtvrtého tutoriálového kroku (červený blok v první řádce) je opět několikanásobně delší než u prvního uživatele a vzorového scénáře. Také si všimneme, že ve scénáři stačily tři logické akce týkající se translační matice ke splnění úkolu (fialové bloky ve druhé řádce pod tutoriálovým úkolem). První i druhý uživatel však provedli akcí pět.



Obrázek 5.13: Čtvrtý krok vlastní úlohy zobrazen v aplikaci Log Viewer

Nejdříve se zaměříme na prvního uživatele. Na Obrázku 5.14 vidíme detail čtvrtého kroku ve vzorovém scénáři. Pokud to porovnáme s detailem čtvrtého kroku prvního uživatele na Obrázku 5.15, uvidíme, že první uživatel nejdříve

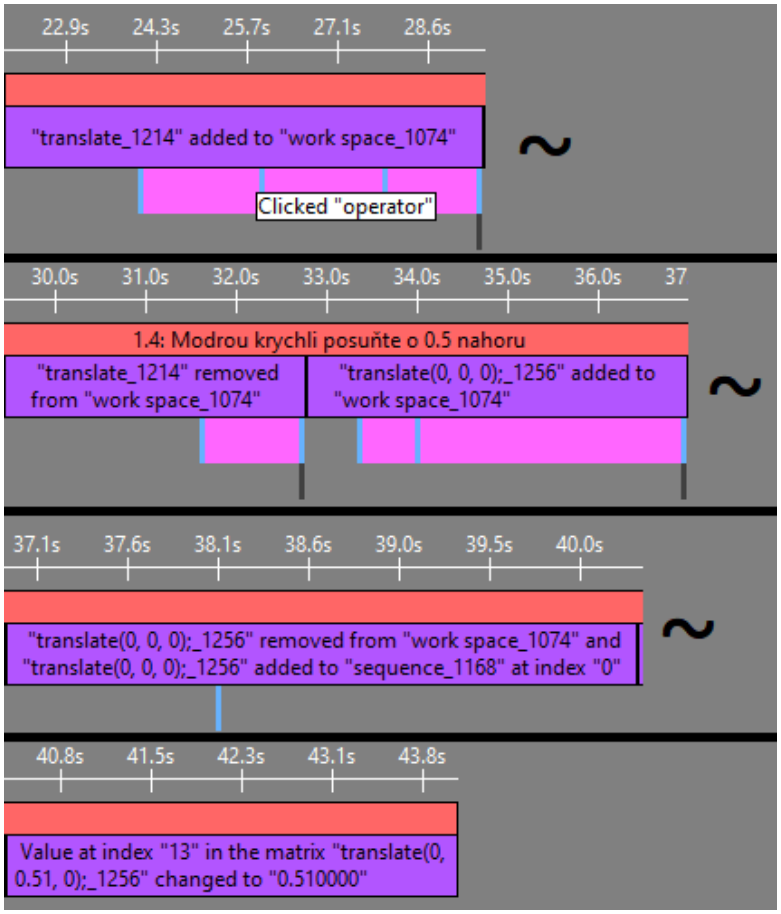
přidání matice hledal v menu operátorů, proto jeho akce přidání matice obsahuje o jedno kliknutí více než vzorový scénář. Na následujících akcích vidíme, že matici, kterou právě přidal, vymazal, hned poté přidal z rozbalovací nabídky novou. Tu následně vložil do sekvence a nastavil hodnotu ve správném poli matice na hodnotu 5.1, místo 5.0. Pravděpodobně se přehlédl a tuto chybu ignoroval. Uživatele jsem se zeptal, proč matici odebral a poté znovu přidal stejnou. Odpověděl, že si nejdříve myslel, že se spletl, ale pak si uvědomil, že to bylo správně.



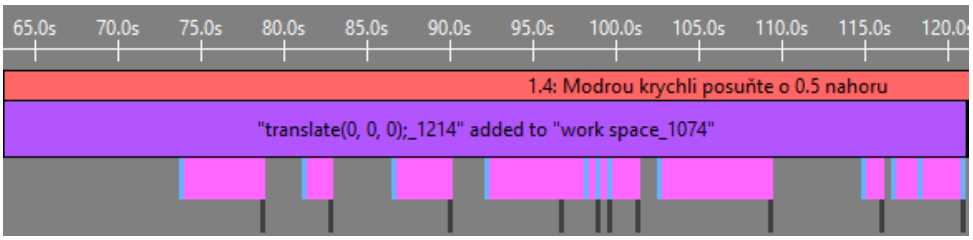
Obrázek 5.14: Tři výřezy po sobě jdoucích událostí čtvrtého kroku ze záznamu vzorového scénáře

Dále se podíváme na záznam druhého uživatele. Na Obrázku 5.16 vidíme, že druhý uživatel měl stejný problém s přidáním matice jako s přidáním krychle v předchozím tutoriálovém kroku. Namísto tří kliknutí v jedné nabídce (viz vzorový scénář na Obrázku 5.14) kliknul uživatel dvanáctkrát v sedmi nabídkách, než se mu podařilo matici přidat do pracovní plochy.

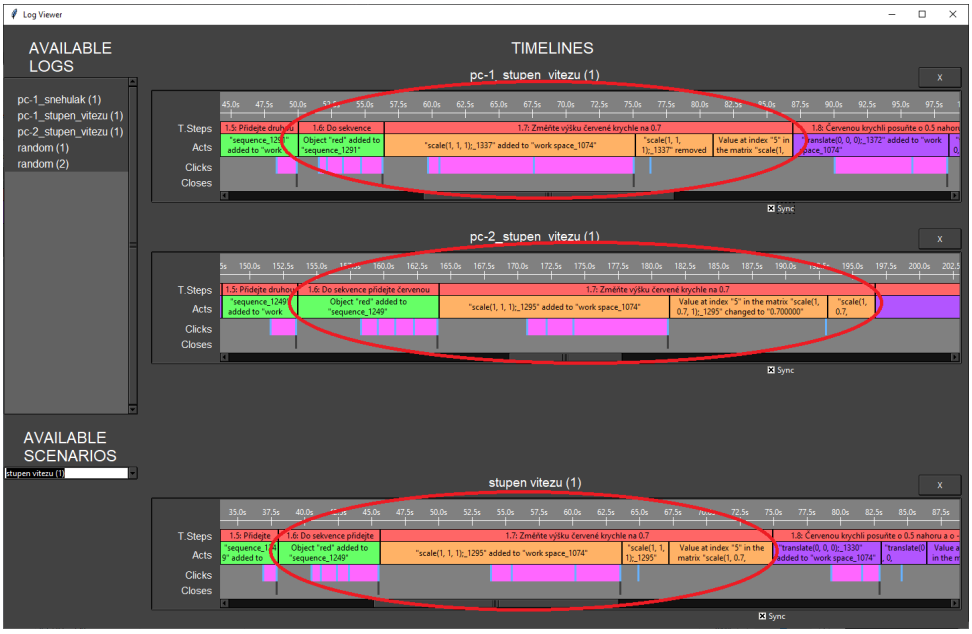
Když druhý uživatel přidával krychli a matici znovu při šestém a sedmém kroku, byl už výrazně rychlejší. Jelikož už věděl kde v nabídce hledat transformační matice, nedělalo mu problém splnit úkol číslo sedm a výšku krychle změnil přibližně stejně rychle jako vzorový scénář a první uživatel (viz Obrázek 5.17). Vzhledem k tomuto zrychlení a velkému počtu „špatných" kliknutí při plnění prvních úkolů můžeme předpokládat, že uživatel má dostatečné znalosti v oblasti teorie transformací, ale pouze se neorientoval v menu aplikace I3T (pravděpodobně kvůli tomu, že neabsolvoval první dvě testovací úlohy).



Obrázek 5.15: Čtyři výřezy po sobě jdoucích událostí čtvrtého kroku ze záznamu prvního uživatele



Obrázek 5.16: Výřez události čtvrtého kroku ze záznamu druhého uživatele



Obrázek 5.17: Šestý a sedmý krok vlastní úlohy zobrazen v aplikaci Log Viewer

5.4 Zhodnocení užitečnosti aplikace Log Viewer

Aplikace Log Viewer pomáhá při manuálním vyhodnocování záznamů získaných z logovacího systému I3T. Díky aplikaci Log Viewer je na první pohled vidět, jak dlouho uživatel pracoval na tutoriálových úkolech a jednotlivých akcích v aplikaci I3T. Získat tyto informace ze samotného textového záznamu by bylo velmi pracné a časově náročné. Log Viewer také umožňuje zobrazit více záznamů najednou, což zjednodušuje analýzu problémových částí záznamů, hlavně při porovnání se vzorovým scénářem. Při porovnávání záznamů je užitečná možnost synchronizace záznamů. Synchronizace se při posunu jedné časové osy na jiný tutoriálový krok pokusí posunout na stejný krok ostatní synchronizované osy a pokud na nich najde tutriálový krok se stejným názvem, nastaví jim stejně velký časový krok.

Logovací systém rozlišuje bloky v I3T (matice, sekvence, operátory) pomocí jejich ID. Aplikace Log Viewer neumožňuje zjistit přesný efekt zaznamenaných změn v těchto blocích (např. ze záznamu změny v poli matice nelze poznat, který objekt je touto změnou ovlivněn). Ovlivněné objekty je potřeba dohledat pomocí ID v předcházejících záznamech. Log Viewer také nelze využít pro analýzu pozice kurzoru, která je zaznemenávána logovacím systémem v I3T.

Kapitola 6

Diskuze

Aplikace na prohlížení zaznamenaných logů slouží k vizualizaci logů a jejich analýze člověkem. Možným dalším rozšířením funkcionality by bylo automatické vyhodnocování uživatelských záznamů oproti vybranému scénáři. Toto vyhodnocení by spočívalo ve vyznačení správných akcí ve správném pořadí a zvýraznění zaznamenaných událostí, které nesouhlasí se vzorovým scénářem v záznamu uživatelské interakce. Vyhodnocení správnosti řešení je však velmi komplikované zejména kvůli velkému množství způsobů, jak jednotlivé úkoly splnit (např. posunutí objektu lze realizovat pomocí vložení matice do sekvence a následné změny pole v matici, provedením těchto akcí v opačném pořadí, nebo pomocí připojení operátoru).

V aplikaci Log Viewer byla implementována možnost synchronizace časových os podle tutoriálových kroků. Při pohybu po synchronizované časové ose dochází k posunutí na stejný tutoriálový krok a synchronizaci časové jednotky pomocí míry přiblížení (tzn. 1 sekunda má stejný počet pixelů v obou časových osách). Předmětem diskuze by mohly být další možnosti synchronizace. Namísto synchronizace na stejně velký časový krok lze synchronizovat „velikost" tutoriálových kroků, zvětšením či zmenšením míry přiblížení tak, aby trvání tutoriálových kroků převedené do pixelů bylo stejné.

Zobrazení otevřených rozbalovacích nabídek je v Log Vieweru omezeno pouze na *je právě otevřeno* a *není právě otevřeno*. Užitečné by také bylo zobrazit hloubku rozbalovací nabídky, ve které se právě uživatel nachází. Ve stávající verzi logovacího systému jsou logy o otevírání a zavírání rozbalovacích nabídek ve stejné logovací kategorii jako klikání na tlačítka v rozbalovacích nabídkách. První možností implementace této funkcionality by bylo vytvoření seznamu tlačítek, která otevírají další submenu (extrémně pracné a náročné na údržbu). Druhou (lepší) možností by bylo vytvoření nové logovací úrovně pro otevírání a zavírání nabídek. Logy z této kategorie poté zpracovávat v Log Vieweru jiným způsobem než logy o klikání na tlačítka v rozbalovacích nabídkách.

Prostor pro zlepšení poskytuje způsob zpracování logu o otevření a zavření rozbalovací nabídky, který v současnosti využívá klíčová slova *open* a *close*. Tato implementace je nyní dostačující. Kdyby ovšem došlo k rozšíření logovaných událostí a i jiné záznamy by obsahovaly tato slova, mohl by zde nastat problém a tyto záznamy by nebyly správně vyhodnocovány. V tomto případě

by se nabízelo zvolit „chytřejší" přístup, například využití regulárních výrazů ke kontrole logu oproti definicím logovaných zpráv.

Potenciál má logování pozice kurzoru. Logovací systém v současnosti sice zaznamenává periodicky pozici kurzoru, ale Log Viewer s ní nepracuje, protože jsme se zaměřili na zobrazení dokončených událostí v aplikaci. V další fázi by bylo zajímavé vytvoření algoritmu, který by barvu záznamu o probíhající logické události upravil podle zaznamenané vzdálenosti kurzoru od „cíle".

Obdobného principu by se dalo využít k navádění uživatele ke konkrétnímu ovládacímu prvku už při samotném plnění úkolu v aplikaci I3T. Uživatel by viděl průběžně aktualizující se nápovědu typu *samá voda, přihořívá, hoří.* Otázku k dalšímu testování je, jestli by tento typ interakce byl názornější než jiná forma zvýraznění ovládacího prvku.

Logovací systém rozlišuje v záznamech jednotlivé obdélníkové ovládací prvky jednotlivých funkčních bloků a editovatelných polí (*Taby*) podle jejich identifikátoru (ID). Užitečným rozšířením logovacího systému by bylo nahrazení tohoto ID pro člověka čitelnější informací (např. sekvence identifikována jménem či typem objektu, který je k ní připojen, matice podle objektu, který ovlivňuje atp.). Návrh algoritmu, který pro každý *Tab* určí vypovídající identifikátor je určitě možným předmětem dalšího výzkumu.

Kapitola 7

Závěr

Bakalářská práce měla za cíl vytvořit logovací systém, tedy knihovnu funkcí, které zaznamenávají události v aplikaci I3T do souboru a interaktivní prohlížeč událostí, který události prezentuje uživateli. Proto má bakalářská práce dvě hlavní části. První částí bylo vytvoření logovacího systému uživatelské interakce pro aplikaci I3T sloužící k výuce transformací. Druhou částí práce bylo vytvoření aplikace Log Viewer, která pomůže nasbírané logy vyhodnotit tím, že je graficky zobrazí.

Součástí první části práce byla analýza požadavků pro sledování uživatelské interakce. V rámci této analýzy bylo zjištěno, že sledování lze rozlišovat pro dva účely - sledování kvůli vývoji rozhraní (zajímáme se hlavně o jednotlivá kliknutí v aplikaci) a sledování kvůli optimalizaci vzorových úloh a výuky teorie transformací (zajímame se hlavně o logické události v aplikaci). Na základě těchto účelů byly určeny čtyři logovací úrovně - sledování pozice kurzoru, klikání na tlačíka v menu, sledování změn hodnot v polích matic a sledování logických událostí v aplikaci.

V aplikaci I3T byl prozkoumán způsob zpracování událostí. Pro aplikaci I3T byl vytvořen logovací systém založený na sledování událostí v aplikaci, který podporuje výše zmíněné úrovně logování. Úrovně lze kombinovat podle cíle, který při logování sledujeme. Logovací systém využívá logovací knihovnu Spdlog, která byla zvolena na základě analýzy požadavků. Na základě průzkumu zpracování událostí byl na vhodných místech v kódu logovací systém integrován do aplikace I3T.

Druhou částí práce bylo vytvoření aplikace pro zobrazení logů - Log Viewer. Pro aplikaci byl zvolen programovací jazyk Python a knihovna pro tvorbu grafických rozhraní TkInter. Log Viewer zobrazuje záznamy z interakce v I3T pomocí časových os. Aplikace dovoluje zobrazit dva záznamy a vzorový scénář najednou pro vzájemné porovnávání. Paralelně zobrazené záznamy lze také synchronizovat na základě logů o plnění tutoriálových kroků.

Po implementaci obou částí byl navržen postup vytváření a vyhodnocování scénářů. V případě sledování za účelem vývoje rozhraní je potřeba se zaměřit hlavně na „špatná" kliknutí, zatímco při sledování za účelem optimalizace tutoriálových úloh a výuky teorie transformací je potřeba se zaměřit na „špatné" logické události v aplikaci.

Logovací systém a Log Viewer byly následně otestovány na třech vzorových

úlohách. Pro vzorové úlohy byly vytvořeny scénáře (ideální sekvence kroků ke splnění lekce). Při testování se jako velmi užitečná ukázala možnost zobrazení těchto scénářů paralelně se záznamem testovaného uživatele. Grafické zobrazení záznamu spolu s možnosti porovnání záznamů umožní testujícímu na první pohled poznat, co bylo u testovaného uživatele za problém.

Logovací systém umožňuje periodicky zaznamenává pozici kurzoru. Tuto informaci však Log Viewer v aktuální verzi nedokáže zpracovat, protože jsme se soustředili na zobrazení dokončených logických událostí v aplikaci I3T. Logování je připraveno na další rozvoj Log Vieweru, který ho bude využívat. Způsob implementace grafického zobrazení logů pozice kurzoru může posloužit jako námět pro další práci.

Logovací systém společně s aplikací Log Viewer bude využit pro snadné získání a vyhodnocení dat týkajících se používání I3T. Tato data plánujeme využít k dalšímu vývoji aplikace I3T, k vývoji výukových tutoriálů a v neposlední řadě k testování znalostí studentů při výuce transformací.

Literatura

[1] Varnika Vayyar, 5 Best User Behavior Tracking Tools for Your Website. *TechJockey.com* [online]. Dostupné z: `https://www.techjockey.com/blog/best-user-behavior-tracking-tools`

[2] Google LLC, Analytics Tools & Solutions for Your Business. *Google Analytics* [online]. Copyright © 2020 Google LLC [cit. 30.04.2020]. Dostupné z: `https://marketingplatform.google.com/about/analytics/`

[3] František Rajtmajer, Co jsou Google Analytics. *nazakladedat.cz* [online]. Dostupné z: `https://nazakladedat.cz/co-jsou-google-analytics/`

[4] Flurry, The World's Most Adopted App Analytics. *Mobile App Analytics Platform for Android & iOS* [online]. Copyright © 2019 Flurry. All Rights Reserved. [cit. 01.05.2020]. Dostupné z: `https://www.flurry.com/`

[5] Medium, Best Tools for Tracking User Behavior on Mobile Apps. *Medium.* [online]. Dostupné z: `https://medium.com/@Appseecom/best-tools-for-tracking-user-behavior-on-mobile-apps-b980cfa2165d`

[6] Microsoft Corporation, Sdílení obrazovky a nahrávání hovorů. *Skype* [online]. Copyright © Microsoft 2020 [cit. 13.05.2020]. Dostupné z: `https://www.skype.com/cs/features/screen-sharing/`

[7] CamStudio, Free Screen Recording Software. *CamStudio* [online]. Copyright © 2005 [cit. 13.05.2020]. Dostupné z: `https://camstudio.org/`

[8] Nvidia Corporation, NVIDIA Nsight Graphics. *NVIDIA Developer* [online]. Copyright © 2020 NVIDIA Corporation [cit. 22.04.2020]. Dostupné z: `https://developer.nvidia.com/nsight-graphics`

[9] Emil Ernerfeldt, loguru: A lightweight C++ logging library. *GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 22.04.2020]. Dostupné z: `https://github.com/emilk/loguru`

[10] Amrayn Web Services, easyloggingpp: Single header C++ logging library. *GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 22.04.2020]. Dostupné z:
`https://github.com/amrayn/easyloggingpp`

[11] Gabi Melman, spdlog: Fast C++ logging library. *GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 22.04.2020]. Dostupné z:
`https://github.com/gabime/spdlog`

[12] Python Software Foundation, Welcome to Python.org *Python Language Reference, version 3.8* [online]. Copyright © 2001 [cit. 14.04.2020]. Dostupné z:
`https://www.python.org/`

[13] Python Software Foundation, pip 20.0.2 documentation. *PyPI - The Python Package Index* [online]. Copyright © 2020 [cit. 22.04.2020]. Dostupné z:
`https://pypi.org/project/pip/`

[14] Python Software Foundation, Virtual Environments and Packages. *Python 3.8.2 documentation* [online]. Copyright © [cit. 22.04.2020]. Dostupné z:
`https://docs.python.org/3/tutorial/venv.html`

[15] Python Software Foundation, tkinter — Python interface to Tcl/Tk. *Python 3.8.2 documentation* [online]. Copyright © Dostupné z:
`https://docs.python.org/3/library/tkinter.html`

[16] Python Software Foundation, tkinter.ttk — Tk themed widgets. *Python 3.8.2 documentation* [online]. Copyright © [cit. 23.04.2020]. Dostupné z:
`https://docs.python.org/3/library/tkinter.ttk.html`

[17] Tkinter Extension Project, ttkthemes: A group of themes for the ttk extenstions for Tkinter. *GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 23.04.2020]. Dostupné z:
`https://github.com/TkinterEP/ttkthemes`

[18] Jamie Matthews, A non-magical introduction to Pip and Virtualenv for Python beginners. *DabApps Blog* [online]. Copyright © DabApps 2020 [cit. 25.04.2020]. Dostupné z:
`https://www.dabapps.com/blog/introduction-to-pip-and-virtualenv-python/`

[19] Object Management Group, About the Unified Modeling Language Specification Version 2.5.1. *Object Management Group* [online]. Copyright © 2020 [cit. 03.05.2020]. Dostupné z:
`https://www.omg.org/spec/UML/`

[20] Tkinter Extension Project, ttkwidgets.TimeLine documentation. *ttkwidgets 0.11.0 documentation* [online]. Copyright © Copyright 2018, ttkwidgets developers [cit. 24.04.2020]. Dostupné z:

https://ttkwidgets.readthedocs.io/en/latest/ttkwidgets/ttkwidgets/ttkwidgets.TimeLine.html

[21] Free Software Foundation, GNU General Public License. *The GNU Operating System and the Free Software Movement* [online]. Copyright © 2007 Free Software Foundation, Inc. [cit. 24.04.2020]. Dostupné z: https://www.gnu.org/licenses/gpl-3.0.html

[22] Neil Young, What is wireframing. *Experience UX* [online]. Copyright © Copyright 2019 Experience UX, 4 Upper Hinton Rd, Bournemouth BH1 2HH [cit. 03.05.2020]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/

[23] Microsoft Corporation, Skriptovací prostředí PowerShell. *Microsoft Docs* [online]. Copyright © Microsoft 2020 [cit. 25.04.2020]. Dostupné z: https://docs.microsoft.com/cs-cz/powershell/scripting/overview?view=powershell-7

[24] Python Software Foundation, Download Python. *Welcome to Python.org* [online]. Copyright ©2001 [cit. 20.05.2020]. Dostupné z: https://www.python.org/downloads/

Příloha A

Manuál k aplikaci Log Viewer

Manuál je rozdělen do dvou částí. První část vysvětluje, jak program nainstalovat a připravit ke spuštění. Druhá část popisuje jak s programem pracovat po jeho instalaci. Manuál je zaměřen na operační systém Windows.

A.1 Instalace programu

Log Viewer je napsán v jazyce Python. Jedná se o interpretovaný programovací jazyk, takže program nekompilujeme do spustitelného binárního souboru, ale spouštíme kód přímo pomocí interpreteru. Proto je potřeba mít nainstalovaný Python, minimální verze 3.7. Python můžeme stáhnout na oficiálních stránkách Pythonu [24].

Po úspěšné instalaci Pythonu můžeme využít instalační skript *setup.ps1*. Skript se nachází v kořenovém adresáři aplikace Log Viewer. Na některých systémech je potřeba změnit výchozí práva pro spouštění externích skriptů. Toho lze docílit jednoduše otevřením *PowerShellové* příkazové řádky s administrátorskými právy a zadáním příkazu „`Set-ExecutionPolicy RemoteSigned`".

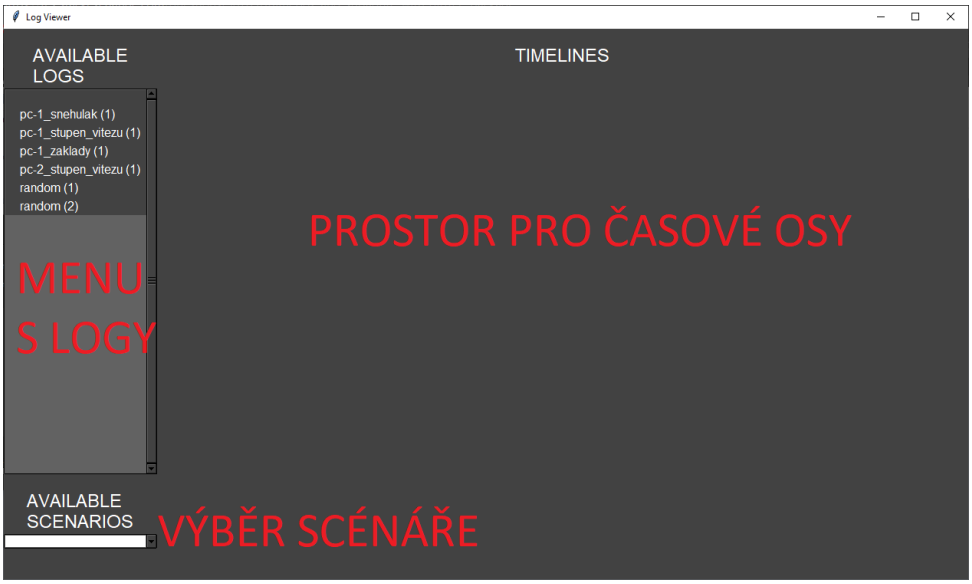
Posledním krokem potřebným před spuštěním programu je nastavení cesty k I3T. V adresáři *log_viewer/config* se nachází soubor *config.py*. Tento soubor otevřeme v libovolném textovém editoru a na posledí řádce souboru upravíme proměnnou *I3T_PATH*. Pozor, zpětná lomítka je potřeba „vyescapovat" jejich zdvojením. Odteď můžeme program spouštět pomocí skriptu *run.ps1*.

A.2 Používání programu

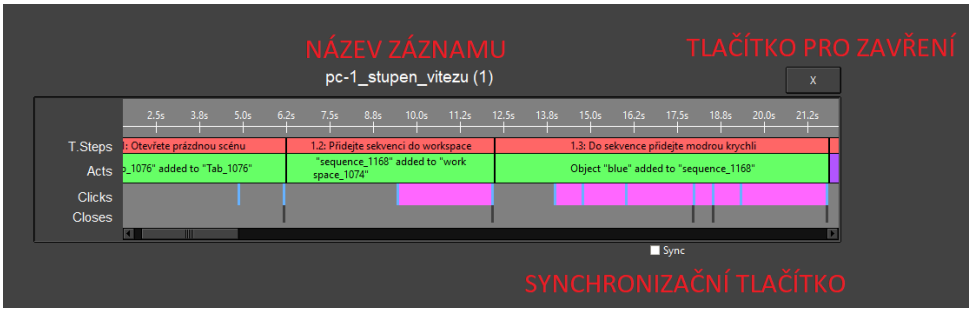
Program obsahuje jednu obrazovku, kterou vidíme na Obrázku A.1. Soubory s logy získané z aplikace I3T je potřeba přesunout do adresáře *logs*. Pokud chceme logy používat jako vzorové scénáře, musíme je přesunout do adresáře *scenarios* (adresáře se nachází v kořenovém adresáři aplikace Log Viewer, viz Obrázek A.3). Jeden soubor s logy může obsahovat několik zaznamenaných seancí (od spuštění aplikace I3T do jejího ukončení) - seance z jednoho souboru budou v menu a v seznamu scénářů odděleny a očíslovány. Adresáře *logs* a *scenarios* obsahují několik záznamů, které byly vytvořeny během testování - ty je možné použít pro vyzkoušení aplikace, nebo je můžeme bez problémů

vymazat. Po přidání souboru s logy není potřeba apliakci restartovat - nalezené logy se aktualizují každých deset sekund (interval lze nastavit v konfiguračním souboru).

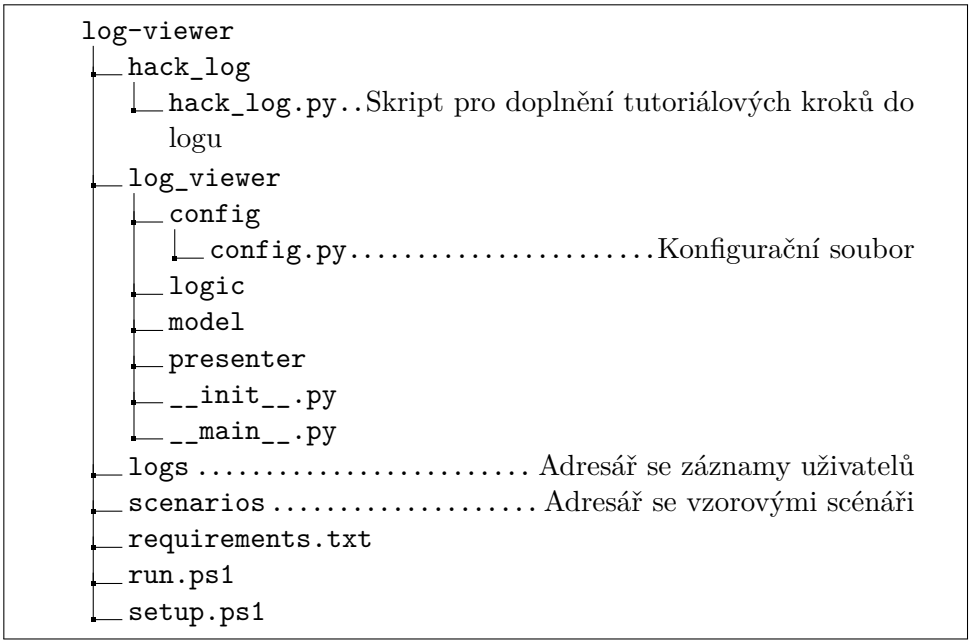
Načtené záznamy v menu s logy či výběru scénářů lze zobrazit pomocí kliknutí na jejich název. Po vybrání se záznam zobrazí v prostoru pro časové osy - viz Obrázek A.2. Časové osy lze synchronizovat pomocí zaškrtávacího tlačítka v jejich spodní části. Časovou osu můžeme zavřít pomocí tlačítka pro zavření (X v pravém horním rohu). Časovou osu lze přibližovat a oddalovat pomocí kolečka myši. Po časové ose se můžeme pohybovat pomocí scrollbaru ve spodní části nebo pomocí shift + kolečko myši.



Obrázek A.1: Prostředí programu s popisem základních komponent



Obrázek A.2: Časová osa s popisem komponent



Obrázek A.3: Struktura adresářů programu Log Viewer

Příloha B

Seznam událostí pro stisknutí a uvolnění tlačítka myši

B.1 onMouseDown

Událost je volána pro nejhlubšího kliknutého potomka. Události můžeme rozdělit do tří skupin:

- Kopírování uzlů v pracovní ploše
  - CameraTransformationForm, MatrixFormBase, OperatorForm, SceneTab, TransformationForm
- Hromadný výběr uzlů v pracovní ploše
  - TransformationSpaceScrollTab
- Otáčení transformačního trackballu v operator > matrix > trackball
  - OperatorOrbitRotateTab

B.2 onMouseUp

Událost je volána pouze pokud tento Tab zpracoval mouseDown - má tedy nastavenou proměnnou „clicked" na true. Proto by se událost měla jmenovat například *onMouseUpOnClickedTab*. Události můžeme rozdělit do osmi skupin:

- Otevření pop up menu pomocí pravého tlačítka myši a přidání do výběru pomocí shift + levé tlačítko myši
  - CameraTransformationForm, OperatorForm, SceneTab, TransformationForm
- Otevření pop up menu pomocí pravého tlačítka myši
  - MatrixFormBase, NumberBox, MatNumberBox, QuatNumberBox, TransformationSpaceScrollTab

- Zavření neinteraktivního tabu (help > about)
  - AbInfoTab, AboutForm
- Otevření webové stránky nebo mailu - text v help > about
  - Label
- Různé eventy (add, delete, open submenu atp.) pro každý button
  - Button, CheckBox, RadioButton
- Kliknutí pravým tlačítkem myši na vstup ve všech operátorech přidá odpovídající operátor a zapojí ho
  - OperatorCurveTab
- Kliknutí pravým tlačítkem myši otevře okno pro úpravu textu (description, hint)
  - GlyphTab, TextTab
- Zobrazení nápovědy - kliknutí levým tlačítkem myši na modrý otazník
  - GlyphTab
- Ovládání operátoru float cycle - play, pause, stop, step next prev tlačítka
  - GlyphTab

B.3 onPassiveMouseUp

Událost je volána, pokud je tlačítko myši na jiném Tabu, než bylo stisknuto (ukončuje se tím tažení myši z jiného Tabu). Název této události by měl být změněn na *onMouseUp*. Události jsou následující:

- OperatorCurveTab - pokusí se napojit tažený spoj
- TransformationForm (sekvence) - přidá taženou matici do sekvence
- TransformationSpaceScrollTab (pracovní plocha) - ukončí hromadný výběr

Příloha C

Definice logovaných zpráv

Formát logovaných zpráv je specifikován v souboru *events_definition.json*. Soubor se nachází v adresáři *resources*, který je v hlavním adresáři aplikace I3T. Obsah souboru vidíme na Obrázku C.1.

```
{
"version" : "1.2",
"mousePos" : "Mouse position: [{}, {}]",
"mouseClick" : "{} mouse button pressed at: [{}, {}]",
"mouseRelease" : "{} mouse button released at: [{}, {}]",
"openPopUp" : "P: Right clicked \"{}\" to show the pop up
↪  menu",
"closePopUp" : "P: Clicked elsewhere to close the pop up
↪  menu \"{}\"",
"closePopUpIn" : "P: Clicked in the pop up menu \"{}\" to
↪  close it",
"button" : "P: Clicked \"{}\"",
"disconnect" : "L: Disconnecting \"{}\" from \"{}\"",
"connect" : "L: Connecting \"{}\" output to \"{}\"
↪  input",
"tabAddIndex": "L: \"{}\" added to \"{}\" at index
↪  \"{}\"",
"tabAdd" : "L: \"{}\" added to \"{}\"",
"tabRem" : "L: \"{}\" removed from \"{}\"",
"objAdd" : "L: Object \"{}\" added to \"{}\"",
"objRem" : "L: Object removed from \"{}\"",
"matrix" : "M: Value at index \"{}\" in the matrix \"{}\"
↪  changed to \"{}\"",
"tutorial" : "T: {}.{}: {}",

"logInit" : ">>> Main logger initialized! <<<",
"mouseLogInit" : ">>> Mouse logger initialized! <<<",
"logEnd" : ">>> Main logger ending! <<<",
"mouseLogEnd" : ">>> Mouse logger ending! <<<"
}
```

Obrázek C.1: Obsah souboru s definicemi logovaných zpráv - *events_definition.json*

Příloha D

Tutoriálové úlohy I3T

I3T Tutoriál

Struktura tutoriálu

- Celý tutoriál se dělí do tematických **Lekcí**, přičemž **lekce** má cca 10 krátkých **Kroků.**
- Každý **krok** se skládá z:
    - Názvu
    - Vysvětlení (zeleně)
    - Úkolu (modře)
    - *Instrukce pro splnění úkolu (kurzivou v podbodu modře)*

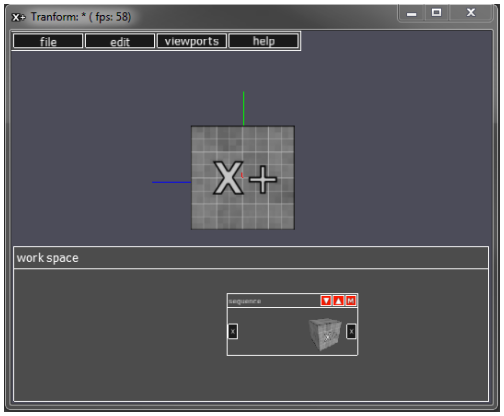
Lekce 1: Můj první objekt

Seznámíte se s uživatelským rozhraním a ovládáním programu. Vytvoříte svůj první objekt a nastavíte úhel pohledu ve 3D scéně.

1. **Workspace**
    - Workspace je plocha v dolní části obrazovky, kde vytváříte a upravujete jednotlivé objekty, matice, sekvence a další entity. Zatím je prázdná.
    - Podívejte se na workspace
2. **Scene**:
    - 3D scene je plocha v horní části obrazovky, kde se zobrazují 3D modely poskládané ve workspace. Scene vidíte nyní také prázdnou, jen se třemi úsečkami, které reprezentují osy X, Y a Z a počátek světové soustavy souřadnic.
    - Podívejte se na scene
3. **Programové menu**
    - Menu se nachází v levém horním rohu. Skrze menu můžete spouštět jednotlivé lekce, nastavovat pohledy na scénu (Viewports) nebo třeba vytvořit zcela novou scénu.
    - Podívejte se na menu
4. **Vytvoření objektu**
    - Objekty reprezentují 3D modely, které budete modifikovat pomocí transformací a sekvencí.
    - Vložte objekt v podobě bílé krychle:
        - *Klikněte pravým tlačítkem myši (dále R-click) na workspace a ve vyskakovacím menu add ➤ vyberte položku sequence.*
        - *Klikněte pravým tlačítkem myši na krabičku sequence a ve vyskakovacím menu vyberte bind object / basics / cube / white*
5. **Otočení úhlu pohledu**
    - 3D scénu si můžete různě natáčet, abyste získali lepší úhel pohledu na svou práci.
    - Otočte 3D scénu:

        - *Nad 3D scénou klikněte a držte stisknuté pravé tlačítko myši. Zároveň táhněte myší (dále R-drag). Uvidíte, jak se vložená krychle postupně otáčí.*

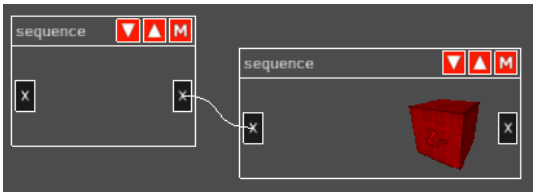
6. **Přiblížení a oddálení**
    - 3D scénu můžete také přibližovat a oddalovat, abyste se v ní lépe orientovali.
    - Přibližte a oddalte 3D scénu:
        - *Najeďte kurzorem nad 3D scénu a jednoduše otáčejte kolečkem myši. Uvidíte, jak se vložená krychle postupně přibližuje a oddaluje. Pokud se krychle nemění, klikněte nejprve na plochu scény, abyste získali fokus.*
7. **Posunutí 3D scény**
    - Poslední užitečná změna pohledu na 3D scénu je její posunutí.
    - Posuňte 3D scénu:
        - *Nad 3D scénou klikněte a držte stisknuté prostřední tlačítko myši. Zároveň táhněte myší (M-drag). Uvidíte, jak se vložená krychle postupně posouvá.*
8. **Viewports**
    - Směr pohledu na 3D scénu můžete měnit také skrze záložku viewports v programovém menu. V nabídce můžete přesně nastavit pohled podél os světových souřadnic (view) a nebo zároveň posunout počátek světové soustavy souřadnic doprostřed scény (world). Kdyby vše z 3D pohledu zmizelo, zachrání vás volba center, která posune těžiště objektů do středu 3D scény.
    - Nastavte úhel pohledu podle světové osy X
        - *Klikněte na záložku viewports v programovém menu a poté vyberte world X*
9. **Heuréka!**
    - První lekce je zdárně u konce. Teď je třeba ověřit, zda se vše podařilo.
    - Zkontrolujte výsledek
        - *Na 3D scénu byste se měli koukat podél světové osy X, kterou tedy ve scéně nevidíte. Na 3D scéně by se měla vyskytovat bílá krychle v libovolné pozici a dvojice souřadných os – zelená osa Y a modrá osa Z.*



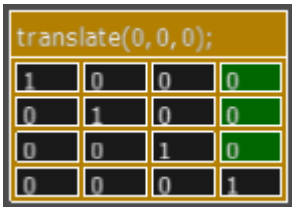
Lekce 2: Základy transformací

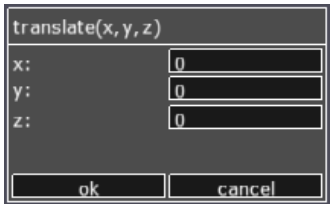
Vytvoříte svůj druhý objekt a aplikujete svou první transformaci. Dozvíte se také o tom, jak transformace upravovat a jak se transformace reprezentují pomocí matic.

1. **Druhý objekt**
   - Kouzlo programu je v hraní si s transformacemi. Abychom viděli, co transformace dělají, musíme je aplikovat na nějaký objekt, nejlépe každou transformaci na jiný.
   - Přidejte do scény druhou krychli, třeba červenou
     - *Klikněte pravým tlačítkem myši (dále R-click) na workspace a ve vyskakovacím menu add ➢ vyberte object / basics / cube / red.*
   - Podívejte se na workspace
     - *Na ploše workspace máme teď dvě krabičky s objekty - jednu pro bílou a jednu pro červenou krychli.*
   - Podívejte se na 3D scénu
     - *Jak to, že tam červená krychle není?*
2. **První transformace - příprava obalu na matici**
   - Červená krychle je na přesně stejném místě, jako ta bílá. Bílá se kreslila jako první a proto není červená vidět. Musíme ji posunout někam jinam.
   - Přidejte do workspace prázdný obal na matice, který se jmenuje sequence a připojte ji před krabičku s červenou krychlí.
     - *Klikněte pravým tlačítkem myši (dále R-click) na workspace a ve vyskakovacím menu zvolte sequence.*
     - *Z pravého křížku na sekvenci vytáhněte myší bílý drát a zapojte jej do křížku vlevo na krabičce s červenou krychlí.*
   - Zkontrolujte výsledek

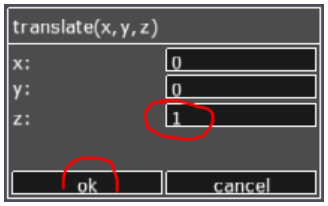


3. **První transformace - vložení matice**
   - Aby matice začala ovlivňovat objekt, musíme ji vložit do zapojené sekvence.
   - Přidejte do workspace transformaci posunutí.
     - *Klikněte pravým tlačítkem myši (dále R-click) na workspace a ve vyskakovacím menu transformation ➢ vyberte translation.*
     - *Stanou se dvě věci: 1) na ploše workspace se objeví žlutá krabička s nadpisem translate(0,0,0); a 2) nad 3D scénou okno s počátečním posunutím v osách x, y a z.*

1) 

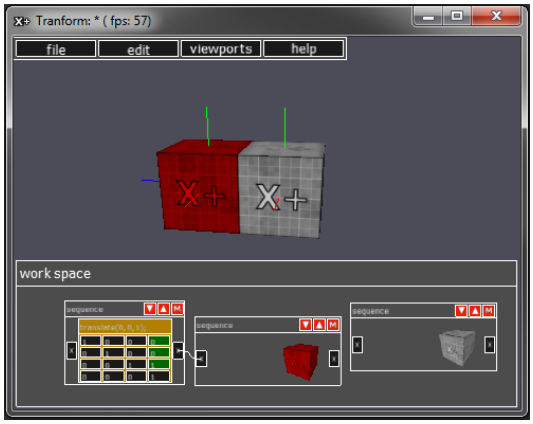
2) 

- *Přepište nulu na jedničku v políčku z: a klikněte v něm na ok.*



- *Uchopte myší žlutou krabičku s nadpisem translate(0,0,1); a přetáhněte ji do připravené a zapojené sekvence (L-drag).*

4. **A je to**
   - Druhá lekce je skoro za námi. Pro jistotu ověřte, že se stále daří, jak má
   - Zkontrolujte výsledek
     - *Měli byste vidět zhruba toto: Na 3D scéně dvě krychle vedle sebe a ve workspace tři krabičky.*



5. **Změna hodnot transformace**
   - Transformace měníme tak, že upravujeme zeleně podbarvená čísla v matici. Máme tři možnosti:
     - Zapsáním hodnoty z klávesnice
     - Výběrem z tabulky
     - Interaktivně myší
   - Posuňte červenou krychli ještě o 1 doleva zapsáním hodnoty z klávesnice
     - *Krátce klikněte myší na třetí políčko v posledním sloupci (L-click) matice ve žluté krabičce a změňte hodnotu z 1 na 2*
   - Vraťte ji zpátky
     - *Klikněte na stejné políčko pravým tlačítkem (R-click) a ve vyskakovacím okně set… vyberte 1*
     - *Zkuste vybrat -1*
   - To nejzajímavější nakonec - hýbejte krychlí interaktivně
     - *Klikněte na stejné políčko opět levým tlačítkem, držte jej stlačené a pohybujte myší nahoru-dolů, nebo doleva-doprava (L-drag)*
     - *Zkuste takto měnit i ostatní zelené hodnoty a koukejte, co to s krychlí udělá*
6. **Shrnutí transformace posunutí**
   - Posunutí se nastavuje v maticí translate hodnotami ve čtvrtém sloupečku.
   - První hodnota posunuje ve směru osy X (té červené)
   - Druhá hodnota ve směru osy Y (zelené) a třetí ve směry osy Z (modré)

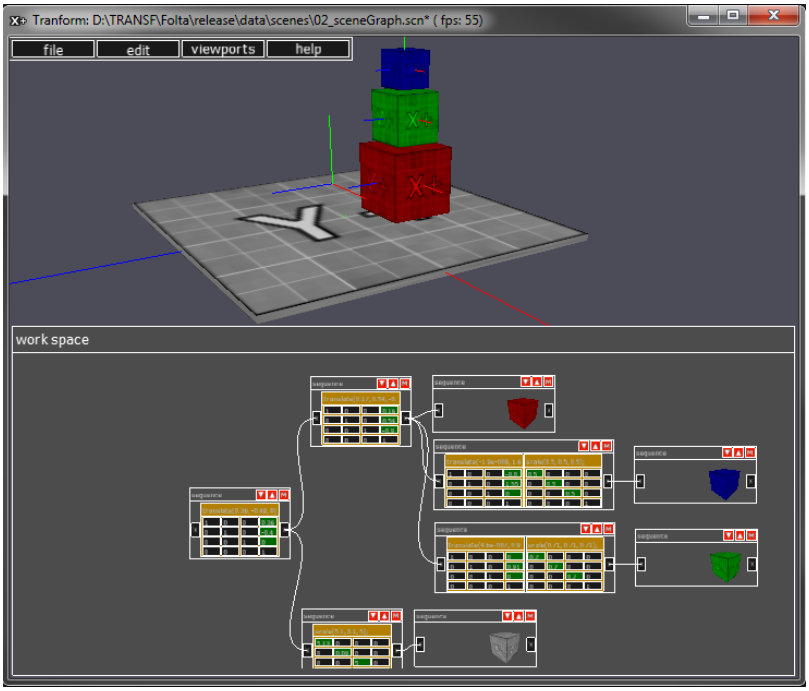
7. **Další druhy transformací a jejich matic**
    - Další transformace se skrývají v menu pod položkou transformation. Stále jsou reprezentovány maticemi 4x4 (o čtyřech řádcích a čtyřech sloupcích), jen se liší políčka, která transformaci v matici definují - a tím pádem je můžeme měnit.
    - Vyzkoušejte si další transformace
        - *Klikněte pravým tlačítkem myši (dále R-click) na workspace a ve vyskakovacím menu transformation ➢ vyzkoušejte postupně transformace:*
            - *scale*
                a. *uniform scale*
                b. *scale*
            - *add rotation*
                a. *euler Angle X, Y a Z*
                b. *rotate*
        - *Potvrďte počáteční hodnoty ve vyskakovacím okně nad 3D space a vložte matici do sekvence. Předchozí matici z ní vyndejte tažením levým tlačítkem myši (L-drag) za horní lištu matice.*
        - *Sledujte přitom, co transformace udělá s objektem, který na ni zprava připojen (nebo je napravo od ní v sekvenci).*
    - Ostatní transformace se týkají dalších logických částí řetězce transformací používaných v počítačové grafice a probereme je později. Slouží např. k nastavení kamery (lookAt) a nastavení projekční transformace (ortho, perspective, frustrum)
8. **Závěrečný úkol**
    - Vyzkoušíme, zda jste všechno z této lekce pochopili. Vytvořte věž ze tří krychlí.
    - Ze tří krychlí postavte věž
        - *Vložte do scény tři krychle: modrou, bílou a tyrkysovou.*
        - *Poskládejte je na sebe - podél osy y.*
        - *Pozor na pořadí transformací, musí být translate – scale.*

Lekce 3: Základy skládání transformací

V této lekci se seznámíte s tím, jak transformace skládat za sebou do sekvence, nebo jak propojovat sekvence do grafu scény. Z matematického pohledu jde o násobení matic ve správném pořadí.

1. **Řazení transformací**
    - Transformace lze za sebe řadit dvěma způsoby. Buď je vkládáme ve správném pořadí do sekvence (matice se násobí ve stejném pořadí, v jakém jsou vloženy), nebo tvoříme hierarchii transformací (propojujeme sekvence do grafu).
    - V menu file / open spusťte dialog pro otevření souboru a v adresáři <adresář s programem i3t> / data / scenes) vyberte soubor 02_sceneGraph.scn.
2. **Vkládání matic do sekvence**
    - Pokud chceme transformovat jen jeden objekt, vložíme transformační matice do jediné sekvence
    - Posuňte a zatočte modrou krychlí
        - *Myší měňte hodnotu v některém ze zelených políček matice eulerAngleY - krychle se otáčí okolo svislé osy Y. Hodnoty v matici jsou vzájemně provázané (obsahují cos a sin úhlu otočení), proto se zároveň mění i v ostatních zelených políčkách.*
        - *Myší měňte hodnoty v matici translate() - krychle se bude posunovat podle vybrané osy.*
    - Vyndejte jednu z matic ze sekvence
        - *Myší uchopte matici za horní okraj a přesuňte ji na plochu.*
        - *Když vyjmete matici rotace okolo Y (eulerAngleY), modrá krychle se otočí zpět do výchozí polohy.*
        - *Když vyjmete matici posunutí (translate), pootočená krychle spadne na plochu na úroveň červené krychle.*
        - *Když vyjmete obě matice, modrá krychle splyne s červenou a není vidět.*
3. **Pořadí transformací je důležité**
    - Pořadí transformací v sekvenci je důležité, protože reprezentuje násobení matic a to není komutativní. Vše si můžeme snáze představit, když čteme složenou transformaci odzadu. Když označíme rotaci písmenem R a translaci písmenem T, znamená pořadí TR posunutí otočené krychle (krychle se otáčí kolem své osy a pak je posunuta na jiné místo) a pořadí RT otočení posunuté krychle podle počátku soustavy souřadnic (jako by krychle byla posunuta po klacíku a ten se otáčel i s posunutou krychlí okolo počátku).
    - Ve stavu TR zatočte krychlí
        - *Posuňte modrou krychli o 1 v ose Z - tj. do bodu (0,1,1).*
        - *Myší měňte hodnoty v matici otáčení.*
        - *Krychle se otáčí okolo své osy.*
    - Prohoďte pořadí rotace a translace v sekvenci připojené k modré krychli - RT
        - *Myší uchopte jednu z matic a přesuňte ji.*
        - *Myší měňte hodnoty v matici otáčení.*
        - *Krychle obíhá jako planeta okolo osy Y ve vzdálenosti 1.*

4. **Hierarchie transformací**
    - Pokud potřebujeme několik objektů transformovat jako celek, využijeme druhý způsob řazení transformací s využitím propojování sekvencí. Transformace působí na všechny objekty k ní zapojené napravo.
    - Když je připojen jeden objekt, působí jen na něj. Když je jich připojeno víc, působí na všechny napravo.
    - Zahýbejte hodnotami matice scale u bílé krychle,
        - *mění se jen bíla podstava.*
    - Zahýbejte hodnotami v matici translate u červené krychle.
        - *posouvají se současně modrá i červená krychle, které jsou od ní doprava.*
    - Zahýbejte hodnotami v matici translate v úplně vlevo,
        - *hýbe se celá scéna, tj. podložka i obě krychle.*
5. **Závěrečný úkol**
    - Vyzkoušíme, zda jste všechno z této lekce pochopili. Vytvořte z této scény sněhuláka ze tří krychlí, který stojí na bílé podstavě a kterým budete umět po podstavě pohybovat.
    - Upravte scénu tak, abyste ze tří krychlí postavili “sněhuláka”
        - *Vložte do scény další krychli – zelenou.*
        - *Nastavte jejich velikost pomocí matic uniform scale na červená* (1)*, zelená* $(\sqrt{2}/2)$ *a modrá* $(0{,}5)$*.*
        - *Poskládejte je na sebe - podél osy y.*
        - *Pozor na pořadí transformací, musí být translate – scale.*
        - *Sněhuláka postavte na znak plus (+) na bílé podstavě.*
    - Zkontrolujte výsledek



Lekce 4: Ovládání programu

V této lekci se seznámíte s dalšími metodami, jak rychle program ovládat. Jde o další užitečné funkce pracovní plochy workspace.

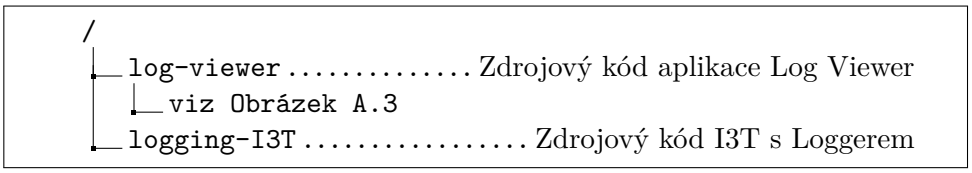
1. **Příprava**
    - Umístěte na pracovní plochu několik krabiček a pospojujte je, nebo využijte krabičky z předchozí úlohy
2. **Přiblížení a oddálení pracovní plochy (zoom)**
    - Celou pracovní plochu můžeme přibližovat a oddalovat, obdobně jako již zmíněnou 3D scénu
    - Přibližte a oddalte celou pracovní plochu
        - *Klikněte na pracovní plochu, abyste ji vybrali (získali fokus). Otáčejte kolečkem myši (mouse scroll). Uvidíte, jak se celá pracovní plocha přibližuje a oddaluje. Všechny krabičky přitom mění svou velikost.*
        - *Bod, na který ukazuje kurzor, zůstává přitom na svém místě.*
3. **Výběr několika krabiček**
    - Chceme-li pracovat s více krabičkami najednou, musíme je vybrat - vybrané krabičky přitom zezelenají.
    - Vyberte krabičky zasažené zeleným výběrovým obdélníkem
        - *Tažením zprava-zdola směrem doleva-nahoru vyberte všechny krabičky, které zasáhne zelený výběrový obdélník.*
        - *Tažení myší musíte začít na volné pracovní ploše.*
    - Vyberte krabičky celé v modrém výběrovém obdélníku
        - *Tažením zleva-shora směrem doprava-dolů vyberte všechny krabičky, které jsou celé v modrém výběrovém obdélníku.*
        - *Tažení myší musíte začít na volné pracovní ploše.*
4. **Posunutí krabiček na pracovní ploše (pan)**
    - Vybrané krabičky se dají po pracovní ploše posunovat. Jedna krabička se posunuje tažením myší za její nadpis (L-drag), vybraná skupina krabiček tažením za kteroukoli část zelené plochy.
    - Označte několik krabiček a posuňte je jinam. Všimněte si, že se propojení krabiček protáhne dle potřeby.
5. **Smazání vybraných krabiček**
    - Jedna krabička se smaže volbou delete v kontextovém menu krabičky.
    - Vybraná skupina krabiček (zezelenalých) se najednou smaže klávesou Del,
    - Označte několik krabiček a smažte je.
6. **Duplikace vybraných krabiček i s propojením**
    - Vybrané krabičky i s jejich pospojováním se dají najednou okopírovat.
    - Stiskněte klávesu Ctrl a přitom klikněte do plochy krabičky nebo vybrané skupiny krabiček *(Ctrl L-click)* - všechny vybrané krabičky se zdvojí a nově vytvořená kopie bude vybrána.
        - *Kliknutím (Ctrl L-click) duplikujete vše, co je vybráno, tažením přesunete nové krabičky na volné místo plochy workspace*
7. **Klávesové zkratky pro pracovní plochu**
    - Při práci ve workspace přibývají moduly a jejich propojení a Vy se můžete začít ztrácet. Mohou se vám hodit následující funkce.
    - Zobrazte všechny krabičky ve workspace
        - *stiskněte klávesu s (malé písmeno “s”).*
    - Označte všechny krabičky ve workspace

        - *stiskněte klávesu a (malé písmeno “a”, jako all).*
    - Invertujte označení krabiček
        - *stiskněte klávesu i (malé písmeno “i”, jako invert).*
    - Vraťte se o jednu operaci zpět
        - *stiskněte klávesu b (malé písmeno “b”, jako back).*
    - Znovu proveďte jednu operaci vrácenou předtím zpět
        - *stiskněte klávesu n (malé písmeno “n”, jako next).*

8. **Teď už snad víte všechno**

Příloha E

Obsah přiloženého CD



Obrázek E.1: Struktura přiloženého CD