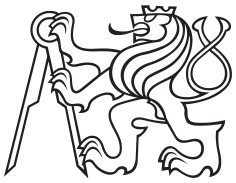
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Grafický editor pro vytváření 3D příběhů

Bc. Oldřich Linhart

Vedoucí: Ing. Jiří Kubišta
Obor: Otevřená informatika
Studijní program: Počítačová grafika
Květen 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Linhart** Jméno: **Oldřich** Osobní číslo: **467111**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Grafický editor pro vytváření 3D příběhů

Název diplomové práce anglicky:

Graphical Editor for 3D Story Telling

Pokyny pro vypracování:

1. Proveďte rešerši existujících systémů pro přípravu interaktivních příběhů.
2. Ve Vámi zvoleném herním enginu vytvořte grafický editor umožňující návrh posloupnosti 3D scén pro vyprávění příběhů. Editor musí umožňovat import 3D modelů, 2D obrazových dat a textových anotací.
3. Při návrhu grafického editoru dbejte na jednoduchost a přehlednost rozhraní.
4. Navrhněte a ověřte v praxi formát pro import/export posloupnosti scén tak, aby bylo možné je použít i v dalších vizualizačních nástrojích pomocí předem definovaného rozhraní.
5. Demonstrujte workflow na ukázkovém příběhu s využitím sady 3D modelů, 2D obrazových dat a scénáře, dodaných vedoucím práce.
6. Vyhodnoťte kvalitu Vámi vytvořeného editoru, srovnejte jej s existujícími nástroji a navrhněte jeho případná další zlepšení.

Seznam doporučené literatury:

1) Liarokapis, F., Voulodimos, A., Doulamis, N., Doulamis, A. (Eds.): Visual Computing for Cultural Heritage. Springer International Publishing, 2020.
2) Arinbjarnar, Maria; Barber, Heather; Kudenko, Daniel: A Critical Review of Interactive Drama Systems. Heslington, York, UK: Games Interactive Entertainment and Drama Group at York. 2009.
3) Europeana Task Force: Europeana as a 'powerful platform for storytelling'. 2021, online. Dostupné z: https://pro.europeana.eu/files/Europeana_Professional/Europeana_Network/European a_Network_Task_Forces/Final_reports/EuropeanaasaPowerfulPlatformforStorytellingReport.pdf
4) Springer, J.; Kajder, S.; Borst, J.: Digital Storytelling at the National Gallery of Art. 2004, online. Dostupné z: https://www.museumsandtheweb.com/mw2004/papers/springer/springer.html

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jiří Kubišta, CESNET, Zikova 4, Praha 6

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

Datum zadání diplomové práce: **10.11.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2023**

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

 © ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Jiřímu Kubištovi za cenné rady a vedení. Dále děkuji všem účastníkům uživatelského testování. Také bych rád poděkoval své rodině a blízkým přátelům za jejich nekončící podporu při mém studiu.

Prohlášení

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

V Praze, 20. května 2022

Abstrakt

Tato práce se zabývá návrhem a implementací grafického editoru, pomocí kterého lze vytvářet digitální interaktivní 3D příběhy. Nejprve jsou analyzovány existující nástroje pro tvorbu interaktivních příběhů. Dále jsou analyzovány druhy interakcí a způsoby, jak je vytvářet. Důraz je kladen na možnosti tvorby interakcí, pro které není nutná znalost programovacích jazyků. Na základě analýzy je vytvořen návrh a následně je aplikace implementována.

Výsledkem práce je aplikace v herním engine Unity, která umožňuje uživatelům import 3D modelů, 2D obrazových dat a textových anotací, ze kterých uživatelé mohou stavět virtuální interaktivní 3D světy.

Klíčová slova: 3D příběhy, vizuální skriptování, Unity, aplikace, grafický editor

Vedoucí: Ing. Jiří Kubišta

Abstract

This thesis focuses on design and implementation of a graphical editor, which can be used to create digital interactive 3D worlds for storytelling. The thesis includes analysis of different types of interactions and ways of creating interactions without previous programming knowledge. On the basis of the analysis, design is created and a working application is developed.

The result of this thesis is an application created in Unity game engine, which allows users to import 3D models, 2D image data and text annotations, which can be used by the users to create virtual interactive 3D worlds.

Keywords: 3D storytelling, visual scripting, Unity, application, graphical editor

Title translation: Graphical Editor for 3D Story Telling

Obsah

Zadání práce iii

1 Úvod 1
1.1 Cíl práce 1
1.2 Struktura práce 1

2 Rešerše a analýza 3
2.1 Interakce 3
2.1.1 Typy interakcí 3
2.2 Vizuální skriptování 4
2.3 Principy návrhu uživatelského rozhraní 7
2.4 Analýza existujících nástrojů . . . 10
2.4.1 Kritéria hodnocení 10
2.4.2 Hodnocení nástrojů 11
2.4.3 Srovnání 20
2.5 Formáty exportu 21

3 Návrh 29
3.1 Persony 29
3.2 Požadavky 30
3.3 Vývojové prostředí 31
3.4 Architektura 31
3.5 Struktura aplikace 33
3.5.1 Editor scén a hierarchie 36
3.5.2 Editor interakcí 42
3.5.3 Zdroje 45
3.5.4 Inspektor 45
3.5.5 Horní lišta 46
3.5.6 Kontextové menu 46
3.5.7 Plovoucí okna 46
3.6 Export 46

4 Implementace 49
4.1 Vývojářské nástroje a externí zdroje 49
4.2 Import souborů 50
4.3 Scéna 51
4.4 Export 51
4.5 Uživatelské rozhraní 51
4.6 Srovnání 55
4.7 Tvorba nových uzlů 57
4.8 Dokumentace 59
4.9 Plánovaná rozšíření 59

5 Uživatelské testování 61
5.1 Metodika testování 61
5.2 Průběh testování 61
5.2.1 Scénář 62
5.2.2 Testování 63
5.3 Výsledky testování 64
5.3.1 Návrh úprav 64

6 Závěr 67

Bibliografie 69

A Seznam použitých zkratek 73

B Zdroje třetích stran 75

C Uživatelská příručka 77

Obrázky

2.1 Příklad skriptů v blokovém skriptovacím jazyce Scratch. 7
2.2 Porovnání textového a vizuálního uzlového skriptu. 8
2.3 Příklad tvorby prostředí v Artsteps. 13
2.4 Příklad spínače ve Warcraft 3 Editoru. 15
2.5 Příklad grafu událostí v Unreal Engine 4. 17
2.6 Příklad skriptu v jazyce CoBlocks. [13] 19
2.7 Příklad 3D scény popsané kódem 2.4 v jazyce VRML. 26

3.1 Komponentový systém Unity. [25] 32
3.2 Skici vybraných částí uživatelského rozhraní. 35
3.3 Příklad rozdílných souřadnicových systémů. [27] 37
3.4 Příklad hierarchie scény. 37
3.5 Zóny pro změnu hierarchických vztahů. 38
3.6 Gizmo manipulátory. [28] 39
3.7 Výpočet úhlu otočení u manipulátoru rotace. 41
3.8 Typy uzlů v grafu událostí. 44

4.1 Barevná paleta aplikace. 52
4.2 Základní grafické prvky v aplikaci. 53
4.3 Příklad komponenty v inspektoru. 53
4.4 Grafická struktura komponenty na obrázku 4.3. 53
4.5 Příklad kontextového menu v aplikaci. 54
4.6 Snímek obrazovky se záložkou scény. 54
4.7 Snímek obrazovky se záložkou grafu událostí. 54

Tabulky

2.1 Srovnání systémů. 21

4.1 Funkcionality aplikace StoryGraph. 56

5.1 Předchozí znalosti testovaných účastníků. 62
5.2 Úspěšnost účastníků při plnění testovacích úloh. 64

Zdrojové kódy

2.1 Příklad formátu BSON. [17] 23
2.2 Příklad formátu JSON. [18] 23
2.3 Příklad formátu XML. [18] 23
2.4 Příklad formátu VRML 2.0. 25
2.5 Příklad dynamických uzlů a jejich propojení ve VRML 2.0. 26
2.6 Příklad XML kódování jazyka X3D. 27
4.1 Příklad datové třídy scény. 52
4.2 Příklad kódu pro vytvoření kontextového menu. . . . 54
4.3 Deklarace třídy uzlu. . . . 57
4.4 Deklarace vstupů a výstupů uzlu. 57
4.5 Příklad inicializační metody uzlu. 57
4.6 Příklad exekuční metody uzlu. 58
4.7 Metody uzlu volané při spuštění a vypnutí příběhu. . . 58
4.8 Příklad spouštění koprogramu uvnitř uzlu. 59

Kapitola 1
Úvod

Příběhy si lidé vyprávěli už od nepaměti. V dnešní době je populární příběhy vyprávět pomocí moderních technologií. Nabízí se otázka, co to vlastně jsou digitální příběhy. Na přesné konkrétní definici se literatura sice neshoduje, ale hlavní myšlenka je obsažena již v názvu – vyprávění příběhů za pomoci digitálních technologií a médií. Ze začátku se příběhy vyprávěly převážně slovně, později se do vyprávění příběhů zapojily další elementy jako zvuky, hudba, herci, fotografie, filmy apod. Moderní digitální příběhy mají ještě jeden mocný nástroj – interaktivitu. Interaktivita umožňuje vzájemnou komunikaci mezi příběhem a jeho diváky.

Samotné příběhy mohou být fiktivní, ale není to podmínka – například dokumentární filmy jsou také forma příběhů. Sledují posloupnost dramatických událostí, ve kterých figurují postavy. Toto se využívá jak v zábavním průmyslu, tak například v informačních, vzdělávacích nebo reklamních kontextech.

1.1 Cíl práce

Tato práce se zaobírá vytvářením digitálních příběhů. Cílem projektu je vytvořit intuitivní editor, který umožňuje návrh, tvorbu a export 3D příběhů, do kterých si uživatelé mohou importovat vlastní 3D modely a 2D obrazová data. Důraz je kladen na jednoduchost a přehlednost editoru. Dalším cílem je navrhnout formát exportu, pomocí kterého bude možné vytvořené příběhy načítat a přehrávat i v jiných vizualizačních nástrojích.

1.2 Struktura práce

Včetně této kapitoly se práce dělí na šest hlavních kapitol.

Druhá kapitola obsahuje analytickou a rešeršní část práce. Ta prozkoumává typy interakcí, které dokážou oživit jinak statické virtuální světy. Dále zkoumá způsoby vizuálního skriptování, které lze využít pro tvorbu interakcí. Poté následuje průzkum hlavních principů pro návrh uživatelských rozhraní. Následuje analýza již existujících systémů, pomocí kterých lze vytvářet digitální příběhy, u těchto systémů se zkoumají jejich silné a slabé stránky.

Konec kapitoly zkoumá existující formáty, pomocí kterých lze popisovat a ukládat interaktivní 3D světy.

Třetí kapitola využívá poznatky z analytické části práce pro návrh vlastní aplikace. Obsahuje návrh uživatelského rozhraní a hlavních systému aplikace. Nakonec je popsáno, jak by měl vypadat formát exportu a co všechno musí obsahovat.

Čtvrtá kapitola popisuje proces implementace. Implementace probíhá dle návrhu ze třetí kapitoly. V této kapitole jsou nejdřív popsány využité vývojářské nástroje. Dále jsou popsány implementační detaily hlavních částí systému a jak se implementace liší od návrhu ze třetí kapitoly. Dále kapitola popisuje srovnání vytvořené aplikace s analyzovanými systémy v první kapitole. Ke konci obsahuje stručný návod, jak lze v aplikaci rozšiřovat možnosti tvorby interakcí. V závěru kapitoly jsou popsány budoucí plány pro další vývoj aplikace.

Pátá kapitola popisuje uživatelské testování, které proběhlo na vytvořené aplikaci a na základě výsledků testování jsou navrženy úpravy.

Šestá kapitola obsahuje shrnutí a závěr celé práce.

Kapitola 2
Rešerše a analýza

V této kapitole je nejprve popsán význam interakcí v digitálních příbězích a s jakými typy interakcí se lze v příbězích setkat. V druhé sekci je uveden pojem vizuální skriptování a jeho význam pro vytváření digitálních příběhů. Třetí sekce se zaměřuje na analýzu existujících nástrojů, pomocí kterých lze vytvářet digitální příběhy. Vybrané nástroje jsou prozkoumány, popsány, ohodnoceny dle předem určených kritérií a vzájemně srovnány. Ve čtvrté sekci jsou uvedeny existující formáty pro ukládání virtuálních scén, které slouží pro následný návrh vlastního formátu pro export vytvořených příběhů.

2.1 Interakce

Jak již bylo zmíněno v úvodní kapitole, moderní digitální příběhy často využívají interaktivitu s diváky, která příběhy odděluje od tradičních forem zábavy, jako je například televize, filmy nebo novely. Interakce umožňuje divákům se přímo podílet na vývoji příběhu. Nejsou to pouze diváci, kteří „konzumují“ dané médium, ale interakce z nich činí přímé účastníky. Umožňuje například prozkoumávat prostředí, ve kterých se příběhy odehrávají, interagovat s obsahem příběhů, či přímo ovlivňovat vývoj příběhů. Je to oboustranná výměna, kde buď obsah reaguje na uživatelské vstupy, nebo obsah vyžaduje nějakou formu interakce od uživatele. [1]

2.1.1 Typy interakcí

Interakce je to, co odlišuje pasivní formu zábavy, kde divák pouze sleduje, poslouchá nebo čte, od aktivní formy zábavy. Interakce se může snažit o pohlcení účastníka v prostředí stimulací co nejvíce lidských smyslů. Stimulovat lze zrak, sluch, hmat a v některých prostředích pro virtuální realitu dokonce i čich. [1]

Lze rozlišovat mnoho typů interakce s digitálním obsahem. Žádné digitální médium nepokryje kompletně všechny typy, avšak následujících 6 základních typů lze dle [1] nalézt ve skoro jakémkoli interaktivním prostředí (dále nazývaném program):

1. *Podnět a odpověď.* Uživatel vytvoří podnět a program na něj zareaguje odpovědí. Podnět může být například kliknutí na nějaký prvek, odpověď

programu může být například přehrání animace nebo zvuku. Podnět může být složen z posloupnosti kroků, které je třeba učinit za sebou, například nejprve uživatel nalezne schovaný klíč a poté s ním otevře zamčené dveře.

2. *Navigace.* Uživatel se může pohybovat v programu a prozkoumávat prostředí. To může být buď otevřenější formou, kde uživatel například volně prozkoumává rozsáhlý 3D svět, nebo více omezenou formou, kde má uživatel na výběr zvolit jednu z více možností v menu nebo se ve scéně pohybovat dle předem definované, omezené navigace.

3. *Kontrola nad objekty.* Uživatel může ovládat virtuální objekty. To může být například přesouvání krabic, otevírání truhel, rozsvěcení baterky apod. Tento typ interakce je sice běžný, ale není univerzální pro každou formu interaktivních programů.

4. *Komunikace.* Uživatel komunikuje s ostatními postavami. Postavy mohou být buď NPC,[1] nebo jiní uživatelé přítomní ve stejném programu. Komunikace může probíhat například v textové podobě, v podobě vybrání možnosti z předdefinovaných dialogů, nebo pomocí hlasu. Stejně jako u třetího typu se nejedná o univerzální typ interakce.

5. *Výměna informací.* Uživatel může posílat informace/data. Tyto informace jsou programem zpracovány, vyhodnoceny a zaslány zpět uživateli. Tento typ interakce se typicky vyskytuje v zařízeních s přístupem k internetu.

6. *Získávání.* Uživatel může získávat věci/informace. Informace mohou být například ve formě faktů), program může umožňovat nákup opravdových nebo virtuálních předmětů, nebo například vlastností (například ve videohrách může získávat nové schopnosti a zbraně pro svoji virtuální postavu). S tímto typem interakce se lze typicky setkat v programech s připojením k internetu a ve videohrách.

2.2 Vizuální skriptování

Interakce mezi uživatelem a virtuálním obsahem lze implementovat pomocí programovacích jazyků. Avšak ne každý tvůrce příběhů bude umět programovat. V následujícím textu bude využíván termín *skriptovací jazyk* a *skriptování*. Rozdíl mezi programováním a skriptováním spočívá v tom, že kód napsaný v programovacím jazyce se překládá do strojového kódu jako celek pomocí kompilátoru, zatímco skriptovací jazyky se pomocí interpretu překládají řádek po řádku – kompilace není zapotřebí a kód lze spustit okamžitě. Skriptovací jazyky jsou podmnožinou programovacích jazyků [2]. Skriptovací jazyky slouží primárně pro rozšiřování a řízení chování existujících aplikací, než pro samotné vytváření aplikací [3].

[1] Non-Playable Character – virtuální nehratelná postava

Mnoho skriptovacích jazyků bylo vytvořeno s myšlenkou udělat programování intuitivnější a jednodušší pro naučení. Rozhraní pro skriptování lze dle [4] rozdělit na **textově založené** a **vizuální**.

Textově založené rozhraní, ve kterém vývojář ručně píše zdrojový kód, který je pomocí interpretu přeložen do podoby, ve které může být na binární úrovni spuštěn. Příkladem rozšířeného textově založeného skriptovacího jazyka je například *Python*,[2] který byl specificky navržen tak, aby byl pro vývojáře snadný ke čtení a pochopení. Z toho důvodu bylo například odebráno mnoho závorek, které se typicky objevují v tradičních programovacích jazycích a místo toho používá odsazení řádků.

Vizuální rozhraní, ve kterém vývojář implementuje funkcionality pomocí vytváření a spojování grafických elementů. Každý grafický element reprezentuje funkci, operaci nebo proměnnou a spojení elementů určuje jejich vzájemné vztahy a tok programu, neboli pořadí, ve kterém se funkcionality jednotlivých grafických elementů vyhodnotí a vykonají. Tato grafická struktura se poté pomocí interpretu opět přeloží do spustitelné formy.

Výhody vizuálního skriptování oproti tradičnímu textově založenému jsou:

- Vztahy mezi různými částmi kódů a proměnnými jsou přehlednější ve formě grafu v rovině [4]. Zároveň je jednoduše pochopitelný tok programu. [5]
- Vývojáři nemusí znát specifickou syntaxi skriptovacího jazyka, ve kterém se funkcionalita vytváří, což pomůže umělcům a designérům, kteří nemusí mít znalosti v oblasti programování [5]. Zároveň může být efektivnější například pro programátory s dyslexií [4].
- Vytváří *sandbox*[3], který, pokud je skriptovací jazyk chytře implementován, omezí možnosti vzniku chyb na minimum. [5]
- Tvůrce skriptovacího jazyka si může přesně určit úroveň abstrakce, kterou jazyk vyžaduje pro plnění svého účelu. Pokud je cíl vytvořit obecný skriptovací jazyk pro všeobecné použití, bude obsahovat veliké množství nízkoúrovňových elementů, pomocí kterých lze složit obecné programy. Pokud je skriptovací jazyk navržen pro specifický projekt, může obsahovat jen relevantní, méně obecné elementy, které jsou vyžadované v daném kontextu a o to jednodušší je pak výsledný systém. [5]

[2] `https://www.python.org/`

[3] Bezpečnostní opatření, které umožňuje uchovávat a spouštět procesy v prostředí odděleném od zbytku systému. Procesy v tomto prostředí mají omezený přístup ke zbytku systému. [6]

Na druhou stranu má vizuální skriptování následující nevýhody:

- Při špatném návrhu skriptovacího jazyka pro konkrétní účel je riziko příliš vysoké abstrakce, což vede k tomu, že i pro vytvoření jednoduchých skriptů je nutné vytvářet rozsáhlé grafické struktury. Proto je nutné si předem pečlivě stanovit, jaké úlohy má jazyk plnit a jaké elementy jsou pro to zapotřebí. [5]
- Vizuální skriptování je méně kompaktní, grafické struktury zabírají více místa. To má za příčinu, že může být náročnější se v rozsáhlejších programech orientovat. Grafy mohou obsahovat mnoho grafických elementů a mnoho spojení mezi nimi, což může být matoucí. [4]
- Pro uživatele s dostatečnou znalostí programování mohou být vizuální systémy příliš limitující a příliš pomalé pro vývoj. [4]

Nejvíce využívané typy vizuálního skriptování jsou skriptování založené na blocích (*Block-scripting*) a založené na uzlech (*Node-scripting*) [4].

Blokové vizuální skriptování

Blokové vizuální skriptování využívá bloky, které na základě jejich tvaru lze na sebe napojovat a tím vytvářet skripty. Bloky mohou značit události, funkce, proměnné a další. Tím, že každý blok má svůj specifický tvar a lze ho napojit pouze v místech kam pasuje, je zabráněno vzniku chyb a zároveň uživateli napovídá, kam lze blok vložit.

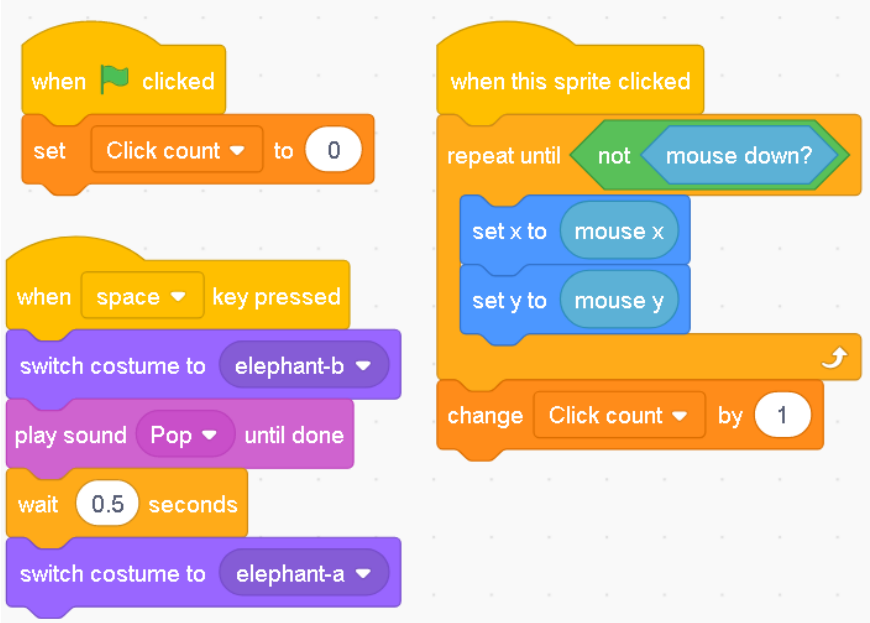
Příkladem blokových skriptovacích či programovacích jazyků jsou například Scratch[4] a CoBlocks[5]. Jednotlivé skripty začínají blokem reprezentujícím událost, která skript spustí. Bloky se skládají vertikálně odshora dolů, čímž je určené pořadí jejich vyhodnocování. Na obrázku 2.1 lze vidět příklad skriptů v jazyce Scratch.

Uzlové vizuální skriptování

Tento typ vizuálních programovacích jazyků využívá uzly, které se spojují pomocí hran/spojů. Samotné uzly mohou reprezentovat funkce, operátory, události, proměnné apod.

Uzly mají vstupy a výstupy. Vstupy a výstupy lze rozdělit na dva druhy: datové a exekuční. Exekuční vstupy a výstupy definují tok, tedy pořadí vyhodnocení uzlů. Datové vstupy a výstupy slouží pro předávaní dat mezi uzly. Z tohoto pohledu lze uzly rozdělit na spustitelné a nespustitelné. Spustitelné obsahují exekuční vstupy. Vyhodnocují se ve chvíli, kdy se k nim dostane spouštěcí signál přes exekuční vstup a datové výstupy se vyhodnotí a uloží. Exekuční výstup může vyslat signál dalšímu uzlu kdykoli během zpracování uzlu. Zároveň uzel může mít více exekučních výstupů (typický příklad může být uzel sloužící pro vyhodnocení podmínky, který výstupní signál větví na

[4] `https://scratch.mit.edu/`
[5] `https://cospaces.io/edu/coding.html`



Obrázek 2.1: Příklad skriptů v blokovém skriptovacím jazyce Scratch.

základě toho, jestli je podmínka splněna). Nespustitelné uzly nemají exekuční vstup. Vyhodnotí se pokaždé, když jiný uzel, který je hranou napojený na datové výstupy tohoto uzlu, vyžádá hodnotu tohoto výstupu. Datový výstup může mít více výstupních hran, ale datový vstup může mít pouze jednu příchozí hranu. Exekuční výstupy a vstupy fungují opačně – tedy jeden exekuční výstup může mít pouze jednu výstupní hranu, ale exekuční vstup může mít více příchozích hran. [7]

Příkladem uzlových vizuálních skriptovacích jazyků jsou například *Blueprint* systém herního engine *Unreal Engine*[6] nebo systém *Visual Scripting*[7] herního engine *Unity*.[8] Na obrázku 2.2 lze vidět příklad skriptu vytvořeného v systému *Visual Scripting* a identickou funkcionalitu vytvořenou v jazyce C#.

2.3 Principy návrhu uživatelského rozhraní

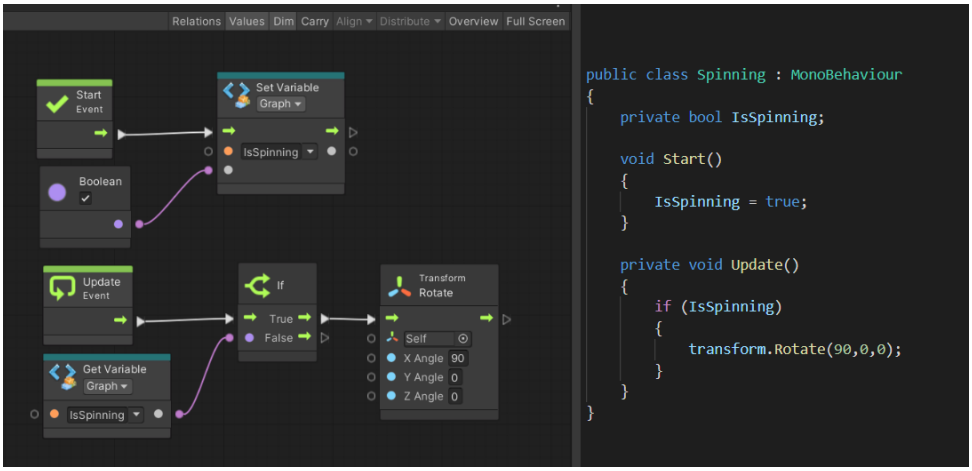
Při návrhu je třeba dbát na principy návrhu uživatelského rozhraní. Je důležité, aby uživatel dokázal s aplikací rychle a efektivně pracovat a aby nebyl z používání aplikace frustrovaný. Obecně lze najít tisíce pokynů a pravidel pro návrh uživatelského rozhraní. Proto v roce 1990 vyvinuli Jakob Nielsen[9] a Rolf Molich sadu pouhých deseti heuristik, které slouží jen jako obecná doporučení a vznikly z pozorování a experimentů [9] [10]. V roce 1994 Nielsen tato pravidla vylepšil na základě dalších analýz a od té doby se

[6] `https://www.unrealengine.com/`

[7] V dřívějších verzích *Unity* byl tento systém dostupný pod názvem *Bolt*, od verze 2021.1 je výchozí součástí *Unity*.

[8] `https://unity.com/`

[9] Odborník na HCI (Human-Computer Interaction) a použitelnost webu.



Obrázek 2.2: Porovnání skriptu vytvořeného pomocí uzlového vizuálního programování s Unity skriptem v jazyce C# s identickou funkcionalitou. [8]

nezměnily. Tyto heuristiky se používají v heuristické analýze, což je jedna z metod testování interaktivního rozhraní. Deset heuristik jakožto doporučení pro návrh uživatelského rozhraní shrnul Nielsen ve svém článku [11] a znějí následovně:

1. **Viditelnost stavu systému**
   Systém by měl být navržen tak, aby uživatelé vždy věděli, v jakém stavu se systém nachází a jaké procesy probíhají, pomocí vhodné a dostatečně rychlé zpětné vazby. Zpětné vazby informují uživatele například o tom, v jaké části systému se nachází, jestli se provádí nějaké operace, jestli může uživatel se systémem interagovat a jestli nenastala chyba.

2. **Propojení systému a reálného světa**
   Měl by se používat jazyk uživatelů, tedy využívat termíny, koncepty a ikony, na které jsou uživatelé zvyklí. Prvky rozhraní by měly být zobrazené v přirozeném a logickém pořadí tak, jako to je v běžném světě. Uživatelé by měli chápat významy termínů bez hledání jejich definic. Chyby či výstrahy by měly být dostatečně zvýrazněny (například červenou barvou). Jazyk uživatelů a jejich mentální modely lze pochopit pomocí uživatelských průzkumů.

3. **Uživatelská kontrola a svoboda**
   Uživatelé by měli mít kontrolu nad tím, co se se systémem děje. Často se děje, že uživatel nějakou akci provede omylem. Mělo by jim být vždy umožněno vykonávaný proces jednoduše zvrátit a vrátit se o krok zpět bez zdlouhavých procesů a negativních dopadů. Nikdy by se neměli dostat do stavu, ze kterého se nelze vrátit. Prvky sloužící pro vrácení by měly být viditelné a srozumitelně popsané.

4. **Konzistence a standardy**
   Významy slov, situací a akcí by se měly držet standardů a konvencí dané platformy a odvětví. Uživatelé by nikdy neměli přemýšlet nad významem. Uživatelé využívají své zkušenosti z používání jiných systémů a mají svá očekávání na vzhled a chování systému. Nedodržení konvencí vede k tomu, že se uživatelé musí učit nové věci, což zvyšuje jejich kognitivní zátěž.

5. **Prevence chyb**
   Systém by měl být navržen tak, aby v něm nedocházelo k chybám. Tedy tato heuristika neřeší, jak systém reaguje, když k chybě nastane, ale jak předejít tomu, aby k chybě vůbec došlo. Systém by měl nabízet pouze možnosti, které nevedou k chybám, případně rozšířit u nebezpečných akcí systém o potvrzovací okna.
   Chyby lze dělit do dvou kategorií: nehody (slips) a chyby (mistakes). K **nehodám** dochází z nepozornosti uživatele, například kliknutí na jiné tlačítko, než uživatel zamýšlel. K **chybám** dochází, když uživatelův mentální model neodpovídá navrženému systému, tedy když uživatel nepochopí, k čemu daná věc slouží nebo jak ji správně ovládat. Předcházet nehodám lze například pomocí vhodných uživatelských omezení (například zamezení tomu, aby do kolonky pro telefonní číslo šlo vkládat písmena). Předcházet chybám lze například pomocí varovných potvrzovacích oken či podpory pro zvrácení posledního kroku (tzv. *Undo*).

6. **Rozpoznání namísto vzpomínání**
   Elementy, akce a možnosti by měly být viditelné tak, aby si uživatel nemusel pamatovat informace přes různé části systému. Potřebné informace pro používání systému by měly být vždy viditelné nebo jednoduše dostupné. Nutnost pamatovat a vzpomínat si zvyšuje kognitivní zátěž uživatele.

7. **Flexibilita a efektivita používání**
   Pro pokročilejší uživatele by měly být dostupné prostředky pro usnadnění a zrychlení používání systému. Tyto prostředky by měly být skryté pro začátečníky. Do toho patří například klávesové zkratky, možnosti přizpůsobení systému na míru a personalizace nabízených možností pro individuální uživatele.

8. **Estetický a minimalistický design**
   Rozhraní by nemělo obsahovat irelevantní nebo zřídka potřebné informace a prvky. Veškeré zbytečné informace odpoutávají pozornost od těch důležitých. Uživatelské rozhraní by mělo být zaměřeno na podporu plnění primárních cílů uživatelů.

9. **Pomoc uživatelům rozpoznat, pochopit a zotavit se z chyb**
   I přes snahu vyvarovat se vzniku chyb, jak je uvedeno v páté heuristice, pravděpodobně nastane situace, že k nějaké chybě dojde. V takovém případě by se chybová hlášení měla zobrazovat formou, které si uživatelé všimnou. Samotné hlášení by mělo uživatele informovat o vzniklém problému řečí, které budou rozumět, tedy neobsahovat pouze chybové kódy. Pokud je to možné, mělo by hlášení obsahovat i informace o tom, jak se ze vzniklé chyby zotavit a jak chybu vyřešit.

10. **Nápověda a dokumentace**

    Předchozí heuristiky cílí na to, aby rozhraní bylo tak srozumitelné, aby nevyžadovalo žádné další vysvětlení či návody. Ale v některých situacích je potřeba uživatelům poskytnout nápovědu či dokumentaci, aby byli schopni splnit své primární cíle. Tato nápověda by měla být stručná, ale zároveň obsahovat veškeré potřebné kroky ke splnění daného cíle. Nápověda by měla být jednoduše dostupná ve chvíli, kdy jí uživatel potřebuje. Do toho patří jak *tooltip*,[10] tak například průvodce, který uživatele provede po uživatelském rozhraní, nebo uživatelská příručka.

2.4 Analýza existujících nástrojů

2.4.1 Kritéria hodnocení

Než bude provedena analýza již existujících systémů pro vytváření digitálních interaktivních příběhů, je nutné si stanovit, dle jakých kritérií budou systémy zkoumány a hodnoceny. Kritéria budou dělena do dvou hlavních kategorií, do kategorie **funkcionality** a do kategorie **uživatelského rozhraní**.

Funkcionalita

Tato kategorie se zaměřuje na „sílu“ systémů pro tvorbu interaktivních příběhů. Tím je myšleno shrnutí všech funkcionalit, které jsou dostupné pro vytváření příběhů a jejich konfigurovatelnost. Pro tuto kategorii byla vybrána následující kritéria:

Import Jaký obsah lze do příběhů přidávat. To mohou být například 3D modely, obrázky, textové anotace, zvuk, animace atd. Lze zkoumat, jaké mají v tomto ohledu systémy omezení. Jaké formáty dat umožňují nahrávat, jestli jsou omezeny svojí paměťovou velikostí, jestli jsou 3D modely omezeny pouze na statické modely nebo lze importovat i animované modely apod.

Vytváření světů Jak lze s importovaným obsahem pracovat při vytváření virtuálních 3D světů a scén příběhů. Jestli si uživatel může pouze vybrat ze seznamu předpřipravených scén, do kterých si doplní objekty, nebo

[10] Textová nápověda, která se uživateli zobrazí při umístění kurzoru přes specifický objekt.

může vytvořit kompletně vlastní svět. Jak složitá je manipulace s obsahem při tvoření scén.

Interakce Jaké jsou v příběhu možnosti interakce dle typů popsaných v sekci 2.1.1. Jakou volnost má uživatel v definování interakcí, jaké funkce jsou uživateli přístupné (například klikání na obsah, stisky kláves, vytváření komplexnějších interakcí pomocí skriptování atd.). Jaké reakce příběhu je možné vyvolat (například přehrání zvuku, přehrání animace atd.) a jaké jsou možnosti navigace kamery po scéně. Jestli je možné vytvářet pouze lineární příběhy nebo se může pomocí interakce diváků příběh vyvíjet nelineárním způsobem.

Export Jestli lze příběhy přímo přehrávat uvnitř systémů, ve kterých jsou vytvořeny a jaké jsou možnosti exportu příběhů ve formě, kterou lze jednoduše zpracovat a zobrazit v jiných systémech.

Uživatelské rozhraní

Tato kategorie je zaměřená na samotné grafické uživatelské rozhraní. Zkoumá, zda jsou dodrženy principy zmíněné v sekci 2.3 a jestli jsou vhodně aplikované v kontextu tvorby interaktivních příběhů.

Jednoduchost Jestli je pro nové uživatele rozhraní dostatečně přehledné a jednoduché, aby se v něm rychle zorientovali a naučili pohybovat. Jestli je z popisků funkcí zřejmá jejich skutečná funkce a jestli jsou uživatelé schopni efektivně najít funkce, které potřebují.

Efektivita Jestli jsou pro pokročilejší uživatele přístupné funkce pro urychlení a usnadnění práce, například klávesové zkratky.

Bezpečnost Jestli je uživatelské rozhraní bezpečné proti chybám, kterých se uživatelé mohou dopustit, jestli je vždy umožněno zrušit poslední akci a jestli je uživatel varován před vykonáním potencionálně nebezpečných úprav.

Hodnotit by bylo možné i pomocí kritérií jako způsob distribuce, cena a podpora operačních systémů, ale to nejsou relevantní faktory pro návrh vlastního editoru a proto nebudou brány při hodnocení v potaz.

2.4.2 Hodnocení nástrojů

Tato podkapitola se zabývá existujícími systémy pro vytváření interaktivních příběhů. Nejdřív je proveden výběr několika relevantních systémů, ty jsou stručně popsány a poté ohodnoceny dle kritérií definovaných v sekci 2.4.1.

Vlastní průzkum ukázal, že ačkoli existuje veliké množství aplikací pro vytváření příběhů, naprostá většina z nich buď neumožňuje pracovat ve 3D, nebo je jejich funkcionalita velice omezená, nebo jsou vytvořeny pro jiná cílová zařízení, například pro mobilní zařízení s pomocí AR[11]. Proto byly

[11] Augmentovaná Realita.

pro hodnocení vybrány i aplikace, které nejsou přímo cílené na vytváření interaktivních příběhů, ale jejich funkcionality to umožňují.

Artsteps

Artsteps[12] je webová platforma, která umožňuje uživatelům vytvářet a procházet virtuální výstavy, například umělecké nebo muzejní 3D galerie.

Uživatel má k dispozici rovnou čtvercovou plochu v prostoru. Nejprve vytvoří zdi a dveře svého virtuálního prostředí. Poté může jednotlivé zdi, podlahu a strop obarvit a potáhnout je texturami. Alternativně si uživatel může vybrat jednu z několika již existujících šablon, ve kterých už ale nemůže upravovat rozložení a vzhled zdí.

Do tohoto prostředí lze následně vkládat objekty. Objekty mohou být obrázky, videa, 3D objekty a text. Pro každý objekt se dá nastavit jeho název, popis, licence, rozměry, jestli je interaktivní (tedy jestli se na něj dá kliknout) a jestli se má při kliknutí přehrát zvuk.

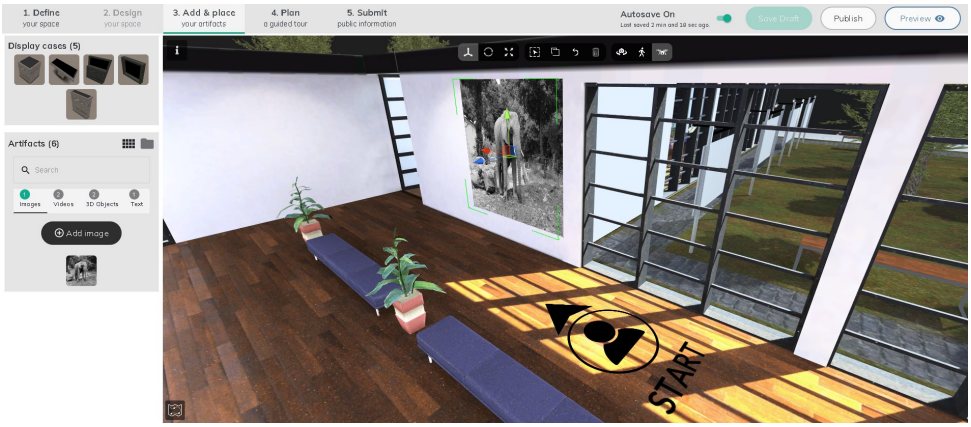
Poté je možné volitelně vytvořit vedenou prohlídku vytvořeného prostředí. V prostředí se určí posloupnost zastávek, každá zastávka má specifikovaný pohled kamery, název a popisek. Mezi těmito zastávkami se pak při prohlídce uživatel může přesouvat.

Pro vytvořenou prohlídku lze nastavit název, kategorii a volitelně poutavý obrázek, popis a audio, které bude hrát po celou dobu prohlídky.

Import Systém umožňuje importovat 3D modely, zvuk (mp3, wav, ogg), obrázky (jpg, png) i videa (mp4, webm). Pro import 3D modelů lze použít kombinace více formátů kvůli texturám a materiálům – například kombinace souborů obj + mtl + textury. Podporován je i modernější formát glb, který sice umožňuje ukládání animací, ale systém s nimi pracovat neumí. Chybí podpora pro formát fbx a formáty používané nejrozšířenějšími programy pro tvorbu 3D modelů (3ds, max, blend, ...). Druhá největší limitace je velikost importovatelných souborů s obrazovými daty, 3D modely a videi, která je omezena na 4 MB. Výhodou je, že se dají objekty importovat i přes externí zdroje – například obrázky přes vyhledávání ze služby *Flickr*, 3D modely přes *PolyAPI* a videa přes *YouTube* odkaz.

Vytváření světů Světy jsou omezeny na rovnou čtvercovou plochu. Zdi se dají vytvářet jen na bodech uniformní mřížky, která pokrývá plochu. Tedy lze vytvářet pouze osově zarovnané a diagonální zdi. Aplikovat texturu a barvu lze pouze na celou podlahu, celý strop a celistvé části zdí ve stejném směru. U dveří nelze upravovat vzhled ani velikost, nastavit lze jen jestli jsou otevřené či zavřené a jestli to může uživatel kliknutím myší měnit. Vlastní obsah se dá vkládat, pozicovat, škálovat, kopírovat atd. U vedené prohlídky nelze měnit pořadí vytvořených zastávek, vertikální pozice zastávek nelze měnit. Příklad tvorby prostředí lze vidět na obrázku 2.3.

[12] `https://www.artsteps.com/`



Obrázek 2.3: Příklad tvorby prostředí v Artsteps.

Interakce Uživatel se může chůzí pohybovat po prostředí. Jediná interakce s vloženým obsahem se spouští kliknutím na objekt, což zobrazí vyplněné informace o objektu, případně přehraje zvuk a zpřístupní ovládací prvky například pro ovládání videa. Pokud je v prostředí současně více uživatelů, mohou vzájemně vidět svoji pozici a komunikovat spolu přes textový chat. Vložené 3D modely jsou statické a nepodporují přehrávání animací. Uživatel se může pohybovat mezi určenými zastávkami. Cesta mezi zastávkami je určena automaticky a pokud jsou v cestě například zavřené dveře, uživatel se o ně zasekne.

Export Příběhy lze přehrávat a testovat přímo ve webovém prostředí. Hotové příběhy lze sdílet přímo na webové platformě, kde je lze najít dle kategorií, názvu, návštěvnosti, hodnocení ostatních uživatelů apod. Lze je i sdílet přes odkaz nebo vkládat na externí webové stránky přes HTML.[13]

Jednoduchost Uživatelské rozhraní je velmi intuitivní. Prvků není na obrazovce nikdy moc a jsou vždy aktivní jen ve správném kontextu. Pro každou jednotlivou část tvorby příběhu je dostupný návod. Po scéně se lze pohybovat třemi způsoby pohybu kamery: obíhací kamera, pohyb ve třetí osobě a volně ovladatelná kamera, což umožňuje jednoduchou a intuitivní manipulaci s importovaným obsahem.

Efektivita Pro důležité funkce jsou dostupné klávesové zkratky pro urychlení práce. Klávesové zkratky jsou v kontextu dalších aplikací pro vytváření 3D scén většinou konzistentní. Nevýhodou je zdlouhavý proces při importování obsahu. Každý vkládaný objekt musí mít vyplněný titulek a rozměry a objekty lze vkládat pouze po jednom.

Bezpečnost Vždy je přístupné tlačítko pro vrácení poslední akce. U nebezpečných akcí, které ovlivní například celou scénu, je potvrzovací okno. Zároveň systém provádí automatické ukládání příběhu každou minutu.

[13]Hypertext Markup Language – hypertextový značkovací jazyk.

Shrnutí. Systém umožňuje vytvářet jednoduché scény s importovanými objekty, ve kterých se uživatelé mohou intuitivně pohybovat a scény si prohlížet. Systém podporuje i prohlížení scén ve virtuální realitě. Nevýhodou systému je velice omezená interakce s objekty ve scéně, absence skriptovacího jazyka pro tvoření vlastních interakcí a omezení na statické 3D modely.

Warcraft III World Editor

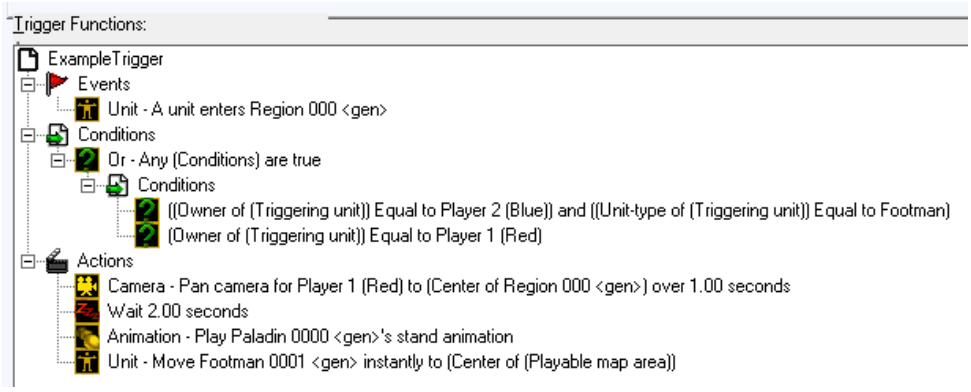
Ačkoli se nejedná přímo o aplikaci pro vytváření interaktivních příběhů, představuje tato aplikace velice silný nástroj pro vytváření vlastních map a kampaní pro hru *Warcraft III*. Tato RTS[14] hra obsahuje kampaně, které mají formu interaktivních příběhů. Uživateli je vyprávěn příběh pomocí kinematografických scén, kde spolu postavy kampaně mluví a interagují. Následně je uživateli zadaná mise, kterou se snaží splnit ovládáním svých postav. Cíle misí mohou být například získání předmětu, postavení základny nebo zničení nepřátelské základny. Kampaně obsahují všech šest typů interakcí zmíněných v sekci 2.1.1. Kampaně mohou být tvořeny pomocí tohoto editoru a proto se jedná o relevantní systém, který je vhodné prozkoumat.

Aplikaci lze rozdělit na několik hlavních částí, které jsou relevantní pro vytváření příběhů. Editor terénu, který umožňuje měnit terén mapy, umisťovat budovy, postavy, kamery, regiony a ozdobné objekty. Editor zvuku, který umožňuje uživateli importovat a exportovat zvuky. Editor objektů, pomocí kterého lze upravovat data postav a budov (například jejich rychlost pohybu, cenu za vytrénování/vybudování, velikost, ...). Zároveň umožňuje vytvářet nové vlastní postavy. Editor kampaně, ve kterém lze poskládat několik map v pevně dané posloupnosti do kampaně, definovat text nahrávacích obrazovek, název kampaně, popis atd. Správce importu, který umožňuje uživateli nahrát si vlastní soubory do mapy. To mohou být obrázky, zvuky, 3D modely atd. Důležitou částí je Editor spínačů. Spínač slouží jako skript, který má definované události (kdy se má spustit), podmínky (za jakých podmínek se může spustit), a akce (co se má provést). Přes tyto spínače je možné vytvořit veškerou interakci, kterou hra nabízí. Poslední částí je Správce objektů, který spravuje odkazy na všechny objekty (postavy, předměty, budovy, regiony, spínače atd.) a jejich vzájemné vztahy (které postavy figurují v jakých spínačích apod.). Příklad spínače lze vidět na obrázku 2.4.

Import Aplikace umožňuje importovat vlastní zvuk (wav, mp3, flac), obrázky (tga) a vlastní 3D modely ve specifickém formátu mdx a jejich textury ve formátu blp, tedy podpora pro import vlastních dat není příliš rozšířená.

Vytváření světů Světy jsou omezeny na obdélníkovou plochu nastavitelné velikosti. Pomocí Editoru terénu lze měnit hrbolatost povrchu a vytvářet skály a rokle. Zároveň je možné vytvářet vodní plochu. Na terén je možné rozmísťovat jednotky, budovy a ozdobné objekty, to vše mohou být vlastní importované modely. Na terénu lze dále vytvářet v uniformní mřížce

[14] Real-Time Strategy – strategie v reálném čase.



Obrázek 2.4: Příklad spínače ve Warcraft 3 Editoru.

pomocné regiony, které lze použít v Editoru spínačů pro specifikace lokací různých interakcí. Nakonec lze vytvářet a umisťovat do světa kamery.

Interakce Veškerá vlastní interakce se tvoří v Editoru spínačů. Každý spínač má seznam *Událostí*. Pokud kterákoli událost nastane, spínač se může spustit. Událost může být například uplynutí času, zmáčknutí klávesy uživatelem, vstup postavy do určeného regionu apod.). Spínač se spustí, pokud jsou splněny všechny *Podmínky* z dalšího seznamu. Podmínka může být například specifikace typu postavy, která do regionu vstoupí nebo specifikace vlastníka postavy. Pokud jsou podmínky splněny, vykonají se postupně odshora dolů *Akce* ze třetího seznamu. Akce mohou být cokoli, co hra nabízí. Může být ovládána kamera, postavy, animace, zobrazení dialogu, textu, obrázků a mnoho dalšího. Nevýhodou je, že lze u všech těchto komponent vybrat pouze ze sady předdefinovaných funkcionalit a nelze tedy vytvářet úplně libovolné interakce.

Export Mapy i celé kampaně lze exportovat v souborech specifických formátů. Tyto soubory lze jednoduše importovat do hry *Warcraft III* a v ní je spustit. Limit velikosti vlastní mapy je 256 MB, což zahrnuje veškerá vlastní importovaná data.

Jednoduchost Uživatelské rozhraní je relativně přehledné a uživatelsky přívětivé. Z ikon tlačítek je zřejmá i jejich funkcionalita, při namíření myší na tlačítka se zobrazí jejich funkce a klávesová zkratka. Nevýhodou je mírně nestandardní způsob správy oken. Editor terénu je součástí hlavního okna. Všechny ostatní komponenty se otevírají jako úplně nová, nezakotvená okna. Více prvků v grafickém rozhraní má identickou funkci. Způsob vytváření spínačů pomocí vybírání ze seznamu funkcí a ručním vyplňováním dat je intuitivní a přívětivé i pro uživatele, kteří nemají žádné zkušenosti s programováním, ale postrádá více vizuální reprezentaci.

Efektivita Všechny důležité funkce mají své klávesové zkratky, což umožňuje urychlování práce. Tyto zkratky jsou zároveň konfigurovatelné. Při

vytváření spínačů lze hledat v seznamu funkcí i pomocí textu. Zároveň je pro pokročilé uživatele dostupná možnost psaní spínačů pomocí vlastních skriptů v textovém skriptovacím jazyce JASS.

Bezpečnost Systém umožňuje vracet poslední kroky. U nebezpečných akcí se zobrazuje potvrzovací okno. Pro skriptování pomocí JASS je v systému dostupný validátor skriptů JASSHelper. Lze nastavit i automatické ukládání v pravidelných intervalech.

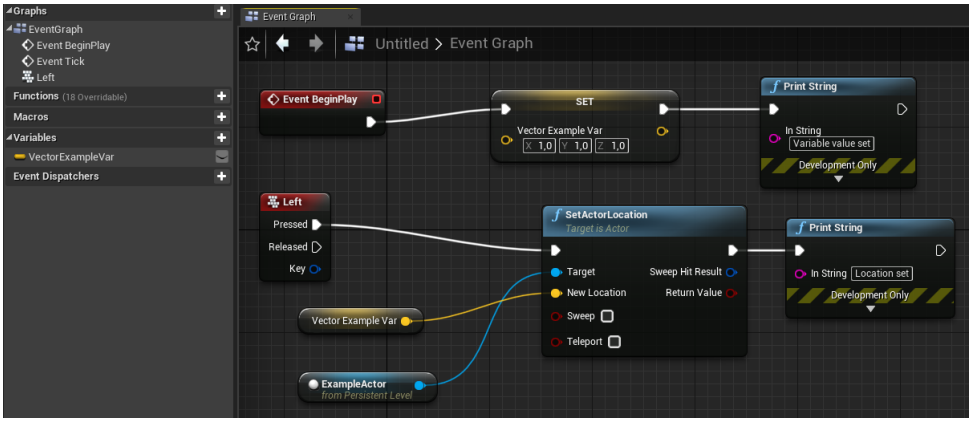
Shrnutí. Systém představuje silný nástroj pro vytváření vlastních příběhů/kampaní. Nevýhodou je, že jsou příběhy a interakce stále limitovány na kombinace elementárních funkcí v samotné hře. Tedy samotný grafický vzhled a herní logika nejsou příliš upravovatelné. Další nevýhodou je, že exportované příběhy lze spouštět pouze přes samotnou hru. Není tedy možné například využít jiné systémy pro vykreslování. Maximální velikost map značně limituje množství a velikost vlastních importovaných dat.

Unreal Engine 4

Unreal Engine 4 (dále jen UE4) je čtvrtá verze herního engine od firmy *Epic Games*. Opět to tedy není systém zaměřený specificky na vytváření digitálních příběhů, ale obsahuje funkcionality pro vytváření herních světů a komplexních systémů interakcí. Jakožto rozšířený herní engine má širokou podporu pro import i export, vytváření virtuálních světů i tvorbu interakcí.

UE4 obsahuje systém uzlového vizuálního skriptování *Blueprint*, jehož princip je popsán v sekci 2.2, pomocí kterého mohou uživatelé vyvíjet funkcionality bez přímé znalosti programování. V systému Blueprint vývojáři tvoří takzvané grafy událostí, obsahující uzly (dále nazývané krabičky) a hrany mezi nimi. Krabičky reprezentují například události, objekty ve scéně, proměnné nebo funkce. Příkladem může být krabička, která posune objekt ve scéně o určitou vzdálenost. Taková krabička má tři hlavní vstupy – exekuční tok určující, kdy se tato akce provede, datový vstup určující objekt, který má být posunutý a třetí vstup vektor určující, o kolik se má objekt posunout. Jako výstup má opět exekuční tok určující, co se má stát po dokončení této akce. Každá scéna (zvaná *Level*) má vlastní globální graf událostí. Zároveň jednotlivé objekty mohou mít svůj vlastní graf událostí. Tyto objekty se dají vkládat do scén, kde vykonávají funkcionality dle grafů událostí. Příklad systému *Blueprint* lze vidět na obrázku 2.5.

Import Systém umožňuje importovat obrovské množství formátů a typů dat. Pro 3D objekty je možné použít formát obj nebo například fbx, který umožňuje importovat celé scény, animace, kostry apod. Zvuk lze importovat například v bezztrátovém formátu flac, dále mp3, wav atd. Obrazová data podporují běžné formáty jako bmp, jpg, png, tga atd. Videa jsou podporována například ve formátech mp4, webm, m4v a wmv. Importovat lze i multimediální toky, fonty, popisné soubory ve formě json a další. Pomocí zásuvných modulů lze importovat i další soubory, například soubory typů CAD, Cinema4D a glTF.



Obrázek 2.5: Příklad grafu událostí v Unreal Engine 4.

Vytváření světů I v rámci vytváření světů je systém velice rozšířený. Kromě standardního vkládání a polohování objektů ve scéně má mnoho pomocných funkcionalit pro snazší polohování a práci s objekty. Oproti předešlým hodnoceným systémům tvoří objekty ve scéně hierarchii, tedy jeden objekt může být potomkem či rodičem jiného objektu. To je velice užitečné při vytváření a manipulaci s komplexními scénami. Zároveň je daleko lépe konfigurovatelné vykreslování objektů. Pro objekty lze vytvořit komplexní materiály, objekty mohou mít další vizuální komponenty jako normálové a spekulární mapy. Uživatel má plnou kontrolu nad osvětlením scény a mnoha dalšími funkcemi pro konfiguraci vykreslování scén.

Interakce Pomocí rozšířeného systému vizuálního skriptování lze vytvářet komplexní interakce. Nevýhodou zde může být to, že je skriptovací systém zamýšlený pro obecný vývoj funkcionalit využívaných v hrách. Takové funkcionality nemusí mít využití v kontextu digitálních příběhů. To způsobuje, že je systém příliš abstraktní, složitý a obsahuje obrovské množství typů krabiček, ve kterých je složité se zorientovat. Výhodou je, že posloupnost krabiček s jedním vstupním exekučním tokem lze spojit do jedné vlastní krabičky reprezentující funkci nebo makro (tedy posloupnost krabiček, ze kterých se skládá).

Export Vytvořený obsah lze exportovat a využívat v dalších projektech ve formě tzv. *assetů*. Celé projekty/příběhy lze exportovat ve formě spustitelných aplikací. Podporováno je mnoho cílových platforem jako Windows, macOS, Linux, iOS, Android, Xbox One a mnoho dalších.

Jednoduchost Největší nevýhoda tohoto systému spočívá v jeho složitosti. Jelikož se jedná o rozšířený herní engine, nabízí obrovské množství funkcionalit, uživatelské rozhraní je složité a často velmi neintuitivní. Naučit se pohybovat v rozhraní bude novému uživateli trvat delší dobu. Mnoho funkcí neposkytuje jednoduše dostupné informace o jejich významu a funkci a uživatel je odkázán na hledání odpovědí v dokumentaci. Zároveň

i samotná dokumentace je často velice stručná a nedostatečná k pochopení složitějších funkcí. Pro vytváření funkcionalit pomocí vizuálního skriptování sice není nutná znalost programování, ale je to velice užitečná znalost, jelikož skriptovací systém využívá běžné programovací koncepty, jako jsou podmínky, cykly, proměnné, volání funkcí apod.

Efektivita Systém obsahuje široké množství konfigurovatelných klávesových zkratek, které umožní pokročilým uživatelům se v rozhraní pohybovat rychleji a efektivněji. Samotný proces tvorby grafu událostí může být pro pokročilého uživatele zdlouhavý, jelikož kvůli vysoké abstrakci musí uživatel i pro vytvoření jednoduchých funkcí většinou vytvořit a nastavit celou řadu krabiček. Pokročilí uživatelé ale ocení možnost vytváření funkcionalit přímo psaním kódu v jazyce C++, kterým lze kompletně nahradit systém vizuálního skriptování. V takovém případě je ale nutná pokročilá znalost jazyka C++.

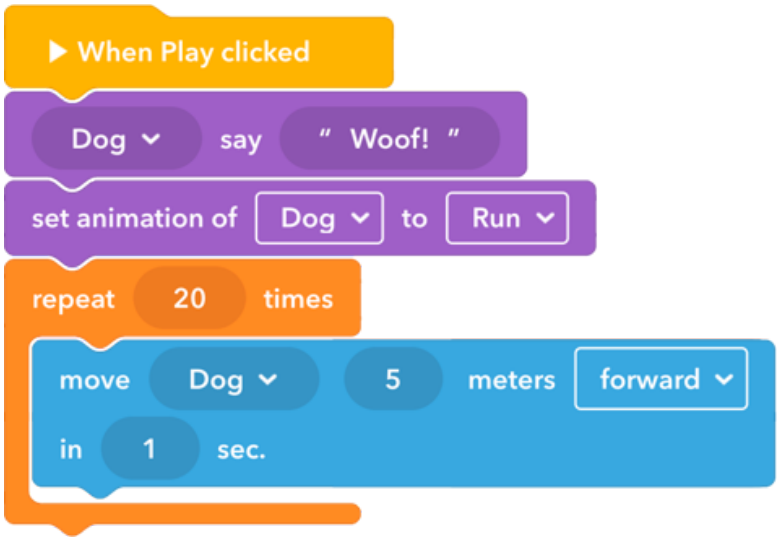
Bezpečnost Uživatel se může vždy navrátit o krok zpět nebo vpřed. Akce, které v daný moment není možné provést, zobrazují varovná hlášení, nebezpečné akce mají potvrzovací okno. Podporováno je jak automatické ukládání, tak přímo integrované verzování přes řadu verzovacích systémů, tedy se na vývoji může podílet současně více lidí. Nevýhodou je, že program není příliš stabilní a není vzácné, že kvůli chybám „spadne“ (ukončí se).

Shrnutí. Systém obsahuje obrovské množství funkcí a možností a uživatel má ve vývoji velikou volnost. To na jednu stranu znamená, že systém umožňuje vytvořit komplexní herní světy a interakce. Na druhou stranu to zároveň znamená, že je systém složitý na ovládání a pro nové uživatele je matoucí. Pro tvoření složitějších interakcí je skoro nezbytné znát základy programování. Možnosti importu a exportu jsou široké.

CoSpaces Edu

CoSpaces Edu [12] je systém cílený na studenty, umožňující vytvářet interaktivní virtuální 3D světy. Slouží primárně k vytváření interaktivních příběhů, výstav, her, simulací a 360° prohlídek. Na vývoji se mohou podílet skupiny studentů pod vedením učitelského profilu. CoSpaces Edu funguje v prohlížeči a zároveň lze stáhnout jako aplikace na počítač, tablet i mobilní zařízení. Vytvořené světy lze sdílet s ostatními a prozkoumávat i ve virtuální a rozšířené realitě.

Projekt je dělený na jednotlivé scény. V každé scéně je k dispozici čtvercová plocha, uvnitř které může uživatel stavět svůj svět. Může také využít nabídky několika šablon prostředí (například město, vesnice, oceán, ...). Do scény lze vkládat 3D objekty, text, obrázky a videa. Systém obsahuje knihovnu předpřipravených 3D objektů, mnoho z těchto objektů obsahují přehrávatelné animace (například model truhly s předpřipravenými animacemi pro otevírání a zavírání). Zároveň si uživatel může importovat své vlastní 3D modely.



Obrázek 2.6: Příklad skriptu v jazyce CoBlocks. [13]

Do projektu lze importovat i zvuky, které lze přehrávat pomocí skriptování. Objekty mají kolizi a lze jim nastavit i působení fyziky.

Systém obsahuje blokový vizuální skriptovací jazyk CoBlocks, jehož princip je popsán v sekci 2.2, pomocí kterého lze programovat interakce ve scénách. CoBlocks podporuje programátorské koncepty jako jsou cykly, podmínky, proměnné, funkce, seznamy apod. Každá scéna má své vlastní skripty, které nelze sdílet napříč více scénami. Příklad jednoduchého skriptu v CoBlocks lze vidět na obrázku 2.6.

Import Do projektu lze importovat vlastní 3D objekty, videa, obrázky a zvuky. 3D objekty lze importovat ve formátech fbx a obj+mtl. Pro importování více objektů současně lze použít formát zip. Systém nepodporuje přehrávání animací pro vlastní importované 3D objekty. Zvuk lze importovat ve formátech mp3, wav, aac a m4a, nebo nahrát přímo v aplikaci mikrofonem. Importovat obrazová data lze ve formátech jpg, png, gif, svg a bmp. Videa jsou podporována pouze ve formátu mp4.

Vytváření světů Pro vytváření světů obsahuje systém jednoduché a přívětivé rozhraní. Objekty se do scény vkládají přetažením z knihovny. Ve scéně tvoří hierarchii a lze je pozicovat, rotovat a škálovat. Systém obsahuje pomocné funkce pro pozicování, například posun a rotace po pravidelných nastavitelných intervalech a automatické přichytávání objektů na jiné objekty. Samotné scény a jejich vykreslování jako například osvětlení jsou jen velmi málo konfigurovatelné. Nevýhodou je, že každá scéna je velikostí limitována na konstantně velikou krychli.

Interakce Systém nabízí hned několik způsobů tvorby interakcí. Blokový vizuální skriptovací jazyk CoBlocks umožňuje vytvářet interakce i bez širší znalosti programování. Pro pokročilejší uživatele systém nabízí i textové programovací jazyky Python a TypeScript. Jazyk CoBlocks

není tak nízkoúrovňový, jako je systém Blueprint popsaný v sekci 2.4.2 a díky tomu je jednodušší se v něm orientovat a pracovat. Na druhou stranu kvůli omezení, že jednotlivé bloky lze skládat pouze vertikálně odshora dolů, má systém menší škálovatelnost a pro větší projekty se stává nepřehledným.

Export Vytvořené projekty nelze přímo exportovat, ale lze je sdílet pomocí vygenerovaných odkazů, QR-kódu nebo veřejně vystavit v galerii projektů přímo v aplikaci. Těmito způsoby sdílení lze příběhy pouze přehrávat, nikoli dále upravovat. Aby se mohlo více uživatelů podílet na vytváření projektů, musí být součástí tzv. třídy, která obsahuje uživatele typu studenti a učitelé. Učitelé v rámci třídy mohou vytvářet zadání úloh, které studenti plní individuálně nebo ve skupinách. Změny ve scénách se zobrazují v reálném čase pro všechny účastníky.

Jednoduchost Uživatelské rozhraní je přehledné a obsahuje pouze nezbytné prvky. Nepoužívané části rozhraní lze kdykoli skrýt. Aplikace obsahuje odkazy na návody, ale přímo v aplikaci není pro většinu prvků dostupný jakýkoli popis či návod. Stavební bloky CoBlocks obsahují lidsky srozumitelný text, pochopitelný i pro mladší studenty.

Efektivita Systém obsahuje pro nejdůležitější funkce nekonfigurovatelné klávesové zkratky. Nevýhodou je, že některé tyto zkratky jsou nestandardní v kontextu tvorby virtuálních světů a většina z nich není uvedena u konkrétních nástrojů a uživatel se o nich dozví až po prostudování on-line návodu. Vizuální skripty, jejich části, objekty i skupiny objektů lze kopírovat. Výhodou je, že pro pokročilejší uživatele jsou pro tvorbu interakcí dostupné textové programovací jazyky Python a TypeScript.

Bezpečnost Změny ve scénách jsou automaticky okamžitě ukládány, takže není možné omylem aplikaci zavřít a ztratit neuložené změny. Uživatel se může vždy vrátit o krok zpět i vpřed. Nezvratné akce jako smazání importovaného souboru vyvolají varovné potvrzovací okno.

Shrnutí. Systém umožňuje importovat vlastní digitální obsah. Nevýhodou je, že nelze importovat vlastní dynamické modely s vlastními animacemi. Vytváření virtuálních světů je díky pomocným funkcím jednoduché, výhodou je uspořádání objektů scény do hierarchie a možnost seskupování objektů do skupin. Vytvořené příběhy lze jednoduše přehrávat na široké škále zařízení, velikou výhodou je i podpora virtuální a rozšířené reality. Další výhodou je, že je systém v aktivním vývoji a na oficiální YouTube kanál[15] jsou přidávány videa představující nové funkce a návody pro práci se systémem. Systém je cílen na studenty a využití ve školním prostředí.

2.4.3 Srovnání

Vzájemné srovnání zkoumaných systémů je shrnuto v následující tabulce:

[15] `https://www.youtube.com/channel/UC6VsnmaKQ9MNRpJbFsIhoGw`

	Artsteps	W3E	UE4	CoSpaces
Import obrazových dat	5/5	3/5	5/5	5/5
• běžné formáty (jpg, png)	✔	✘	✔	✔
• alfa kanál	✔	✔	✔	✔
Import 3D modelů	3/5	2/5	5/5	3/5
• běžné formáty (obj, stl, fbx)	~	✘	✔	✔
• podpora animací	✘	✔	✔	✘
Vytváření světů	1/5	3/5	5/5	3/5
• neomezená plocha	✘	✘	✔	✘
• členění do scén	✘	✔	✔	✔
• vertikální dimenze	✘	✘	✔	✘
Tvorba interakcí	1/5	4/5	4/5	3/5
• vizuální skriptování	✘	✔	✔	✔
• textové skriptování	✘	✔	✔	✘
Export	3/5	2/5	5/5	3/5
• spustitelné v editoru	✔	✘	✔	✔
• standalone aplikace	✘	✘	✔	✘
Efektivita	3/5	3/5	5/5	3/5
• klávesové zkratky	✔	✔	✔	✔
• přizpůsobitelné rozhraní	✘	✘	✔	~
Jednoduchost	4/5	4/5	2/5	4/5
• přehledné/čisté rozhraní	✔	✔	✘	✔
• tooltipy, nápovědy	✔	✔	✔	✘
• průvodce	✘	✘	~	✘
Bezpečnost	5/5	5/5	5/5	5/5
• automatické ukládání	✔	✔	✔	✔
• krok zpět a vpřed	✔	✔	✔	✔
• potvrzovací okna	✔	✔	✔	✔
Celkem	**3,1/5**	**3,4/5**	**4,5/5**	**3,6/5**

Tabulka 2.1: Srovnání systémů.

2.5 Formáty exportu

Serializace je proces převodu libovolně složitých datových objektů (datových bloků v paměti, které mohou obsahovat proměnné, datové struktury, funkce, ...) do sekvence bytů, kterou lze jednoduše ukládat v paměti například jako datový soubor na disku, posílat například přes počítačovou síť a ze které lze

opačným procesem deserializace získat zpět původní datový objekt. [14]

Serializaci lze rozdělit na více typů v závislosti na jejich formátu. Formát je důležitý, protože se od něj odvíjí faktory, jako je složitost dat, potřeba lidské čitelnosti serializovaných dat, rychlost serializace, velikost serializovaných dat atd. V praxi se lze setkat s dvěma hlavními typy: binární formát a textový formát.

Binární Datové objekty jsou převedeny do binárního proudu. Výhodou binární serializace je možnost komprese serializovaných dat a vyšší rychlost serializace. Nevýhodou je, že binární formát není lidsky čitelný a interní detaily binárního formátu nemusí být veřejně dostupné, což znamená že data mohou zpracovat a zrekonstruovat pouze aplikace či platformy, které tento formát a detaily procesu serializace znají. [15]

Příkladem binárních formátů je například BSON (binární JSON), vytvořený multiplatformní dokumentovou databází MongoDB. Další příklad je formát protobuf (protocol buffers), vytvořený společností Google. Příklad dokumentu ve formátu BSON obsahujícího text `"hello":"world"` lze vidět v kódu 2.1. [16]

Textový Datové objekty jsou převedeny do textového dokumentu, který popisuje datový objekt. Výhodou tohoto typu je, že má tvůrce větší volnost v určení formátu tohoto dokumentu. Zároveň je formát lidsky čitelný a serializovaná data lze otevřít a upravovat i v jednoduchém textovém editoru. Další výhodou je to, že s textovými dokumenty lépe pracují verzovací systémy, jelikož je snazší slučování změn než u binárních souborů. Při procesu deserializace není zapotřebí dalších procedur jako je dekomprese a data lze zrekonstruovat na libovolné platformě či aplikaci (pokud je dostupný popis výsledného formátu). Nevýhodou je, že proces serializace je pomalejší a serializovaná data jsou objemnější.

Příkladem textových formátů jsou například XML (Extensible Markup Language), CSV (Comma-Separated Values), JSON (JavaScript Object Notation) a YAML (YAML Ain't Markup Language). Příklad dokumentu ve formátu JSON a stejného dokumentu ve formátu XML lze vidět v kódech 2.2 a 2.3. [16]

Pokud je v zájmu bezpečnost datových objektů (například v případě, že objekty obsahují citlivé informace, nebo data, která by uživatel neměl mít možnost měnit), není ani jeden z těchto typů sám o sobě bezpečný a data by měla být ochráněna jinak (například šifrováním).

Řada aplikací, ve kterých je možné vytvářet datové objekty, nabízí pro ukládání a přenášení těchto objektů oba zmíněné typy. V kontextu vytváření příběhů obsahujících scény s 3D modely může být příkladem program Autodesk Maya, který umožňuje vytvářet 3D modely. Projekty vytvořené v tomto programu lze ukládat do souboru. Tento soubor obsahuje serializovaný objekt obsahující informace o vytvořené geometrii, osvětlení, animaci atd. Uživatel si při ukládání projektů může vybrat formát MB (.mb), který je binární, tedy není lidsky čitelný, ale využívá kompresi pro zmenšení datového objemu. Jeho

interní formát není veřejně známý a tedy výsledný soubor nelze otevřít v jiné aplikaci, než je Autodesk Maya. Druhou možností je formát MA (.ma), který je textového typu. Soubor tedy lze otevřít a měnit například v textovém editoru. Prakticky je možné vytvořit celý Autodesk Maya projekt pomocí textového editoru. Rozhraní tohoto formátu je veřejně dostupné. Dalším příkladem jsou datové soubory popisující 3D scény vyvinuté neziskovým průmyslovým konsorciem Khronos Group. Dostupný je opět jak textový formát glTF, tak binární formát GLB.

```
\x16\x00\x00\x00          // total document size
\x02                      // 0x02 = type String
hello\x00                 // field name
\x06\x00\x00\x00world\x00 // field value (size of value, value,
    null terminator)
\x00                      // 0x00 = type EOO ('end of object')
```

Kód 2.1: Příklad formátu BSON. [17]

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Kód 2.2: Příklad formátu JSON. [18]

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Kód 2.3: Příklad formátu XML. [18]

Jedním z formátů pro reprezentaci interaktivních 3D světů je VRML (Virtual Reality Modeling Language), s příponou „wrl“. VRML je současně jazykem, který je mezinárodní normou ISO[16] sloužící pro popis dynamických virtuálních světů [19]. VRML soubory jsou textového typu. Existují dvě verze jazyka, VRML 1.0, která slouží pro vytváření a prohlížení statických 3D světů a jeho nástupce, VRML 2.0 (také označované jako VRML 97). Druhá verze umožňuje daleko rozsáhlejší popis interaktivních scén nezávisle na cílové platformě. Soubory obsahují kolekci objektů, které se nazývají uzly (node). Existuje veliké množství typů uzlů, uzly mohou reprezentovat například tvar, vlastnost nebo skupinu. Skupinové uzly seskupují jiné uzly do jednoho celku,

[16]Standard definovaný mezinárodní organizací pro normalizaci.

čímž lze reprezentovat celou hierarchickou strukturu scény. Uzly reprezentující tvar definují geometrii objektů, ze které se skládá 3D svět. Uzly reprezentující vlastnost modifikují jiné uzly tím, že jim přidávají vlastnosti. Tato vlastnost může být například rotace, měřítko či textura. [20]

Uzly obsahují informaci o svém typu (například barva, krychle, zvuk, bodové světlo atd.), pole a události. Pole vyjadřují vlastnosti či parametry (například poloměr geometrie koule, difúzní barvu materiálu apod). Parametry mají v rámci uzlu svoji výchozí hodnotu, datový typ a unikátní název/identifikátor. Speciální typ pole lze hrubě přeložit jako odhalené pole (exposed field). Základní pole jsou pro ostatní uzly skryté a nemohou je měnit. Odhalené pole lze rozdělit na tři části – výchozí hodnotu, příchozí událost pro nastavení hodnoty a odchozí událost pro signalizaci o změně hodnoty. Tyto parametry jsou tedy schopny přijímat události svého datového typu a přijetím této události se hodnota změní. Zároveň o své nové hodnotě mohou vyslat v okamžiku této změny událost, na kterou mohou reagovat další odhalené pole jiných uzlů.

Pro spojování uzlů, specificky jejich odhalených polí, slouží takzvané cesty, značené klíčovým slovem „route“. Pomocí příkazu route lze spojit odhalené pole jednoho uzlu s odhaleným polem jiného uzlu. Jeden uzel je vysílacím uzlem, který vysílá událost s informací o změně svého parametru. Druhý uzel je přijímacím uzlem, který na tuto událost reaguje změnou hodnoty svého odhaleného pole.

Mezi typy uzlů patří například senzory, které generují a vysílají události, když jsou splněné určité podmínky. Podmínky mohou být závislé na interakci od uživatele. Existují například časové senzory, které generují události v závislosti na uplynulém čase, geometrické senzory, které reagují na vstup uživatele do určité neviditelné geometrie ve scéně, dotykové senzory, které reagují na vstup ukazatele uživatele do potomků tohoto uzlu reprezentujících geometrii. Senzory viditelnosti, které reagují na to, jestli je definovaná geometrie krychle aktuálně viditelná z pohledu uživatele a další.

Další typy uzlů slouží pro definici animací 3D objektů. Tato skupina uzlů se nazývá interpolátoři (interpolators) a je určena pro lineární animaci pomocí klíčových snímků. Tyto uzly umožňují interpolaci například mezi množinou barev, pozic, normál, skalárních hodnot apod. Lze je například použít i pro vytváření prohlídek, kde je pohyb uživatele řízen po předem definované dráze.

Jiný typ uzlu může reprezentovat skript, který přidává do scény nějakou funkcionalitu/chování. Skripty se spouští na základě událostí. Tyto události mohou být buď závislé na uživatelském vstupu, nebo například vyvolány nějakou změnou hodnoty pole v nějakém uzlu. Po spuštění mohou vykonat výpočty, něco ve scéně změnit a vyslat další události.

VRML dále umožňuje uživatelům vytvářet nové vlastní typy uzlů, které se skládají z množiny dalších uzlů. To umožňuje zapouzdřit množinu geometrie, vlastností a chování do nového uzlu, který je pak možný ve scéně instancovat a vytvářet tak vlastní předdefinované objekty. Tato funkcionalita se označuje slovem prototyp.

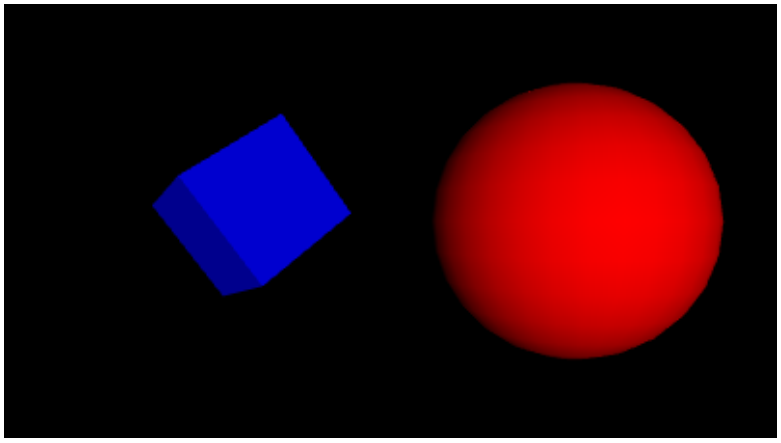
Jiné uzly mohou například aktivovat či deaktivovat detekci kolizí mezi uživatelem a všemi potomky tohoto uzlu reprezentující geometrii, mohou

reprezentovat světelné zdroje ve scéně, mohou spouštět zvuky, reprezentovat mlhu ve scéně a mnoho dalšího. Detailní popis všech typů uzlů a další informace o tomto formátu lze nalézt v oficiální specifikaci VRML. [21]

Příklad jednoduchého VRML 2.0 souboru reprezentujícího 3D svět obsahující červenou kouli, modrou kostku a bodové směrové světlo lze vidět v kódu 2.4. Vykreslenou scénu popsanou tímto kódem lze vidět na obrázku 2.7. Příklad jednoduché části kódu, která předává události mezi uzly a reaguje na dynamickou pozici uživatele ve scéně pro zapínání a vypínání světla a hudby při vstupu uživatele do místnosti lze vidět v kódu 2.5.

```
#VRML V2.0 utf8
Transform {
  children [
    NavigationInfo { headlight FALSE } # We'll add our own light

    DirectionalLight {        # First child
        direction 0 0 -1      # Light illuminating the scene
    }

    Transform {               # Second child - a red sphere
      translation 3 0 1
      children [
        Shape {
          geometry Sphere { radius 2.3 }
          appearance Appearance {
            material Material { diffuseColor 1 0 0 }   # Red
          }
        }
      ]
    }

    Transform {               # Third child - a blue box
      translation -2.4 .2 1
      rotation     0 1 1  .9
      children [
        Shape {
          geometry Box {}
          appearance Appearance {
            material Material { diffuseColor 0 0 1 }  # Blue
          }
        }
      ]
    }

  ] # end of children for world
}
```

Kód 2.4: Příklad formátu VRML 2.0 (převzato z přílohy D viz [21]).



Obrázek 2.7: Příklad 3D scény popsané kódem 2.4 v jazyce VRML. (převzato z přílohy D viz [21]).

```
Group {
  children [
    ... # popis místnosti
    DEF SVETLO SpotLight { on FALSE } # zpočátku světlo nesvítí
    Sound {
      source DEF HUDBA AudioClip { ... }
    }
    DEF AKCE ProximitySensor { ... }
  ]
}
ROUTE AKCE.isActive TO SVETLO.on
ROUTE AKCE.enterTime TO HUDBA.startTime
ROUTE AKCE.exitTime TO HUDBA.stopTime
```

Kód 2.5: Příklad dynamických uzlů a jejich propojení ve VRML 2.0 (převzato z [19] str. 82).

Nástupcem formátu VRML je formát X3D (Extensible 3D), který obsahuje rozšíření formátu VRML. Opět se jedná zároveň o jazyk, který je ISO normou pro popis 3D virtuálních světů. Jazyk VRML rozšiřuje o další možnosti, jako popis rovinných tvarů (2D), podporu NURBS[17], podporu humanoidní animace „HAnim“ nebo třeba částicové systémy. [22]

Jazyk X3D je často nazývaný třetí generací VRML. Nabízí několik druhů kódování/formátu: kódování ve značkovacím jazyce XML (x3d), klasické VRML kódování (x3dv), binární kódování (x3db) a další. Příklad stejné scény popsané ve VRML v kódu 2.4 lze vidět v XML kódování jazyka X3D v kódu 2.6. [23]

[17] Neuniformní kubický racionální B-Spline (aproximační křivky složené z oblouků).

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">

<X3D version='3.0' profile='Interchange'>
  <head>
    <meta name='filename' content='RedSphereBlueBox.x3d'/>
  </head>
  <Scene>
    <Transform>
      <NavigationInfo headlight='false'
        avatarSize='0.25 1.6 0.75' type='"EXAMINE"'/>
      <DirectionalLight/>
      <Transform translation='3.0 0.0 1.0'>
        <Shape>
          <Sphere radius='2.3'/>
          <Appearance>
            <Material diffuseColor='1.0 0.0 0.0'/>
          </Appearance>
        </Shape>
      </Transform>
      <Transform translation='-2.4 0.2 1.0' rotation='0.0 0.707 0.707
          0.9'>
        <Shape>
          <Box/>
          <Appearance>
            <Material diffuseColor='0.0 0.0 1.0'/>
          </Appearance>
        </Shape>
      </Transform>
    </Transform>
  </Scene>
</X3D>
```

Kód 2.6: Příklad XML kódování jazyka X3D (převzato z přílohy C viz [24]).

Kapitola 3

Návrh

Tato kapitola se zabývá návrhem aplikace pro vytváření interaktivních příběhů. Nejprve je třeba určit cílové uživatele aplikace a stanovit si funkční a nefunkční požadavky aplikace. V další části je třeba vybrat technologie a vývojové prostředí, které budou použity pro implementaci. Poté je třeba navrhnout jednotlivé dílčí části aplikace tak, aby aplikace umožňovala plnit určené požadavky. Pro aplikaci byl vymyšlen název **StoryGraph**, který vychází z myšlenky, že aplikace má sloužit k vytváření příběhů.

3.1 Persony

Ačkoli je tvorba digitálních interaktivních příběhů aktivita, která se úzce opírá o herní vývoj a počítačovou grafiku, hlavní myšlenkou v tomto návrhu je to, aby implementovaná aplikace byla přístupná i pro nezkušené uživatele v těchto oblastech. Uživatelské rozhraní by mělo být přehledné a jednoduché. Zároveň by ale aplikace měla být dostatečně flexibilní, aby s ní bylo možné vytvářet rozmanité množství příběhů. Je tedy nutné určit správnou rovnováhu mezi dostatečnou obecností a mezi jednoduchostí.

Na druhou stranu aplikace nemá za účel učit uživatele vytvářet virtuální světy od základů. Proto je v návrhu předpokládaná primární cílová skupina uživatelů taková, která má alespoň základní znalosti v oblasti počítačové grafiky. S touto myšlenkou lze rozdělit cílové uživatele do dvou primárních skupin a na základě těchto skupin vytvořit persony, aby se daly snáze určit požadavky na funkcionality aplikace:

Grafici tvoří skupinu, která od systému požaduje co nejvyšší míru flexibility a volnosti při vytváření příběhů. Chtějí vytvářet rozsáhlé virtuální světy, ve kterých je mnoho různých způsobů interakce s prostředím. Vyznají se v systémech pro vytváření 3D světů a mají zkušenosti i s herními jádry a programováním. V doméně systému se vyznají dobře. Od systému očekávají dostatečné množství funkcionalit a konfigurovatelnost jednotlivých funkcí pro své specifické potřeby. Ocení možnosti, které jim usnadní a urychlí práci v systému, jako jsou klávesové zkratky.

Persona: Ing. Jan Novák je 35letý zaměstnanec v IT firmě. Vystudoval počítačovou grafiku na FEL ČVUT a podílel se na několika graficky

zaměřených projektech pro různé firmy. Jeho firma přijala zakázku na vytvoření virtuální prohlídky muzea, kde návštěvníci mohou muzeem procházet a interagovat s exponáty. Jan musí splnit specifické požadavky zakázky, ve kterých nemá moc flexibility pro úpravu. Potřebuje proto, aby systém nabízel dostatek funkcionalit pro vytvoření virtuálního světa ve vysoké kvalitě a aby v tomto světě šlo vytvořit specifické druhy interakcí.

Storytelling nadšenci je skupina s méně technickými zkušenostmi. Pohybují se ve světě storytellingu a s digitálními příběhy už se setkali. Vědí proto, co mohou očekávat a mají částečnou znalost domény systému. Neumějí nutně programovat, ale mají základní zkušenosti s vytvářením virtuálních světů. Ve své komunitě nadšenců chtějí vytvářet interaktivní příběhy, které mohou spolu sdílet buď pro edukativní účely, nebo čistě pro zábavu. Po systému požadují, aby byl alespoň trochu flexibilní. Zároveň by chtěli, aby byl systém rozšiřitelný a aby vývojáři v případě širšího zájmu komunity přidali nové, žádané funkcionality.

Persona: Jindřich Veselý je 20letý student FIT ČVUT. Před vysokou školou vystudoval technické lyceum v Praze. V minulosti už si vyzkoušel několik systémů pro tvorbu virtuálních světů a jednoduchých příběhů. Hodně se pohybuje na internetu a je součástí online komunity, která vytváří edukativní interaktivní příběhy, které sdílí spolu a s širokou veřejností o nejrůznějších tématech. Na většinu témat mu stačí základní funkcionality systému, ale v některých případech potřebuje typ interakce nebo nastavení specifické pro jeho současný projekt. Je ochoten být v aktivním kontaktu s vývojáři systému a poskytovat nápady a rady pro zlepšení.

3.2 Požadavky

Na základě zadání diplomové práce, analýzy existujících systémů pro tvorbu příběhů a určené cílové skupiny uživatelů byly určeny následující funkční a nefunkční požadavky

Funkční požadavky

F1 Do herního světa je možné vložit vlastní 3D modely, obrazová data a textové anotace.

F2 Hráč příběhu se může po vytvořeném herním světě pohybovat a prozkoumávat jej.

F3 Hráč příběhu může interagovat s vytvořeným herním světem.

F4 Vytváření interakcí je přístupné i pro uživatele bez znalostí programovacích jazyků.

F5 Aplikace umožňuje vytvořený příběh spustit a vyzkoušet.

F6 Vytvořené příběhy je možné vyexportovat v takovém formátu, aby bylo možné je využít i v dalších vizualizačních nástrojích.

Nefunkční požadavky

N1 Aplikace bude mít formu spustitelné aplikace na operačním systému Windows 10.

N2 Uživatelské rozhraní je jednoduché a neobsahuje příliš veliké množství prvků.

N3 Aplikace bude rozšiřitelná pro přidávání dalších typů interakcí a import dalších typů souborů.

3.3 Vývojové prostředí

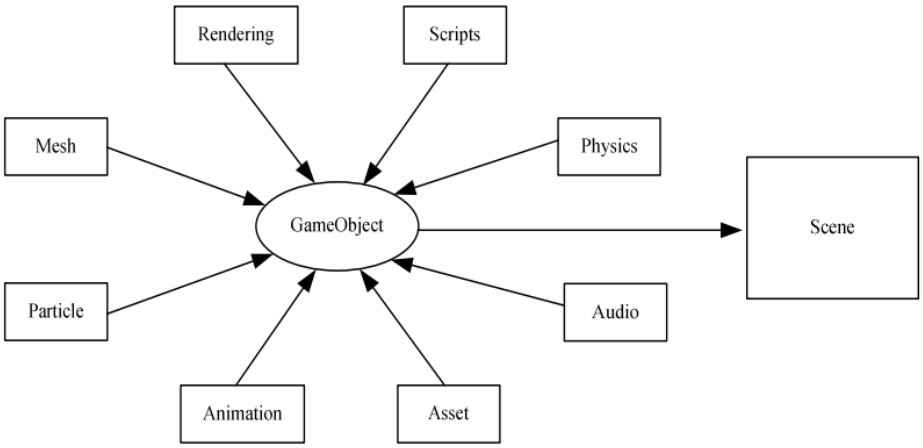
Aplikaci by bylo možné napsat celou samostatně, avšak v kontextu práce s virtuálními světy je výhodnější využít již existující herní engine, aby nebylo nutné vlastnoručně implementovat základní koncepty jako je vykreslování scény, mapování textur na 3D objekty, interní reprezentace objektů ve scéně, elementární prvky uživatelského rozhraní apod. To jsou všechno věci, pro které má většina populárních herních enginů velkou podporu a umožní to soustředit se implementací na konkrétní požadavky aplikace.

Pro vytvoření aplikace StoryGraph byl autorem práce vybrán herní engine **Unity**. Tento engine kvalitně podporuje všechny základní koncepty zmíněné v předchozím odstavci a obsahuje široké množství nástrojů, které dále implementaci usnadní. Dále obsahuje relativně kvalitní dokumentaci a velice aktivní uživatelskou komunitu, která vytváří užitečná rozšíření. Unity obsahuje několik licencí, které se liší na základě příjmu z prodejů, tedy pro účely této aplikace je Unity zdarma. Výhodou je i to, že Unity podporuje široké spektrum cílových platforem, takže by v budoucnu neměl být problém rozšířit podporu pro více operačních systémů. Ve svém základě je ale primárním důvodem volby tohoto enginu fakt, že autor práce má v něm nejširší znalosti oproti ostatním enginům.

3.4 Architektura

Scény či herní světy v Unity jsou založené na komponentovém modelu, který umožňuje vytvářet škálovatelnou architekturu a umožňuje členit herní logiku do komponent, které se vloží na objekt ve scéně a tím upravují jeho chování. Herní svět v Unity je rozdělen na scény.[1] Scény obsahují objekty zvané `GameObject`. Tyto objekty slouží jako kontejnery, na které jsou vkládány komponenty. Komponenta reprezentuje nějakou funkcionalitu či vlastnost objektu, na kterém je vložena. To může být například komponenta `Transform`,

[1] Pro jednoduchost bude v textu předpokládáno, že je v daný okamžik vždy načtena jediná scéna, ačkoli Unity podporuje aditivní načítání scén.



Obrázek 3.1: Komponentový systém Unity. [25]

která slouží pro určení pozice, rotace a škály objektu v souřadnicovém systému svého rodiče. Jiná komponenta může být `Collider`, který reprezentuje kolizní geometrii objektu. Důležitými komponentami jsou uživatelské skripty. Unity má nativní podporu pro vytváření skriptů v jazyce C#. Pomocí skriptů lze vytvářet logiku objektu a pomocí referencí na ostatní komponenty upravovat vlastnosti objektu [25]. Tato struktura je znázorněna na obrázku 3.1.

Jak bylo naznačeno, jednotlivé objekty mohou mít další objekty jako své potomky a mohou být potomkem jiného objektu (pro přesnost, `Transform` komponenta obsahuje tyto informace). To znamená, že objekty ve scéně tvoří hierarchickou strukturu. Hierarchii scény lze reprezentovat jako n-ární strom, kde kořen stromu je samotná scéna.

Další důležitým konceptem je tzv. `Prefab`, který slouží jako šablona, která může obsahovat předdefinované objekty. Tuto šablonu lze ve scénách instancovat, což si lze představit jako vytváření její kopie. Tyto kopie mají přímé spojení se svojí šablonou a pokud se tato šablona změní, změní se současně i všechny její instance (pokud tyto instance konkrétní datovou položku nepřepíšou vlastní hodnotou). To značně usnadňuje efektivitu při návrhu – uživatel může například vytvořit obecnou šablonu pro tlačítko v uživatelském rozhraní, které má stejnou funkcionalitu, jediné, co se liší, je vykonaná akce při stisku a text tlačítka (tedy to jsou hodnoty, které instance přepíšou). Pokud se uživatel zpětně rozhodne změnit vzhled nebo interní logiku stejnou pro všechna tlačítka, stačí upravit pouze šablonu.

Díky možnosti přiřazování skriptů objektům ve formě komponent vede Unity k rozdělení kódu na oddělené moduly/skripty a tím je kód znovupoužitelný. Například každý UI prvek může mít skript, který řídí jeho vzhled a interní logiku. Zároveň je ale typicky nutné, aby spolu jednotlivé prvky a další objekty vzájemně komunikovaly. Tlačítko musí poslat informaci o tom, že na něj uživatel kliknul. Seznam obsahující objekty ve scéně musí získat informaci o tom, že byl nějaký objekt smazán a aktualizovat svůj obsah. Proto je vhodné, aby UI prvky měly své veřejné rozhraní, pomocí kterého lze měnit jejich stav podle potřeby.

Logiku souvisejících skupin objektů je vhodné kvůli přehlednosti a rozšiřitelnosti kódu spravovat přes malý počet centrálních ovladačů. Například všechny objekty v jedné scéně mohou řešit komunikaci přes veřejné rozhraní scény, do které patří. Jednotlivé scény mohou komunikovat přes veřejné rozhraní třídy, která má za úkol spravovat scény. Stejně tak například skupiny tlačítek, kde může být aktivní právě jedno, by měly mít centrální ovladač, který při stisknutí tlačítka informuje všechna ostatní tlačítka o tom, že se musí přenastavit na neaktivní. Kdyby neexistoval tento centrální ovladač, při přidání nového tlačítka do takové struktury by programátor musel manuálně projít všechna existující tlačítka a přidat jim referenci na nové tlačítko, což je z hlediska udržitelnosti čistého a efektivního kódu nepraktické.

Na vzájemnou komunikaci souvisejících objektů se hodí dva návrhové vzory. V softwarovém vývoji značně kontroverzní návrhový vzor `Singleton` se vyskytuje jako třída, která má v celém programu pouze jedinou instanci. Tato instance se uloží do statického atributu, ke kterému lze přistupovat přes veřejnou metodu z jakéhokoli místa v kódu. To vede k tomu, že jednotlivé UI prvky nemusí mít uloženou referenci ani na svůj centrální řídící ovladač. Druhý návrhový vzor se nazývá `Observer`, který reprezentuje vztah mezi objekty, kde jeden objekt je pozorovaný (subject) a jiný objekt je jeho pozorovatel (observer). Jeden pozorovaný může mít více pozorovatelů. Když dojde u pozorovaného objektu ke změně, o které je nutné zaslat informaci, automaticky na tuto změnu upozorní všechny své pozorovatele. V jazyce C# je tento vzor implementován pomocí zpracování a vyvolání událostí (klíčové slovo „event“). [26]

Tím, že jednotlivé související objekty jsou spravovány přes centrální ovladače a jednotlivé ovladače mohou komunikovat spolu, lze zajistit fragmentovanou, škálovatelnou komunikaci všech hlavních systémů aplikace.

3.5 Struktura aplikace

Cílová aplikace je zamýšlena pro počítače, návrh se tedy soustřeďuje na grafické rozhraní pro obrazovky s vyšším rozlišením, ovládané pomocí myši, klávesnice, případně dotykové obrazovky. Aplikace není zamýšlena pro mobilní zařízení.

S výše stanovenou obecnou architekturou aplikace lze navrhnout, z jakých hlavních centrálních systémů se bude skládat a jak vypadá uživatelské rozhraní. Pro rozdělení lze vycházet z funkčních požadavků popsaných v sekci 3.2. Při návrhu uživatelského rozhraní je nutné dbát na principy návrhu popsané v sekci 2.3.

Pro zajištění rozšiřitelnosti aplikace lze centrální systémy a jejich případné uživatelské rozhraní rozdělit do dvou kategorií – centrální a doplňkové. Centrální systémy se starají o hlavní funkcionalitu celé aplikace a jsou pro funkčnost aplikace nezbytné. Doplňkové systémy nejsou pro aplikaci zásadní, ale slouží pro zjednodušení ovládání aplikace, nastavení uživatele a zaručení bezpečnosti.

Kdyby se postup uživatele seřadil chronologicky v situaci, kdy chce vložit do virtuálního světa nový model, postup by mohl vypadat následovně: import dat do projektu → vkládání objektu do scény → tvorba virtuálního světa (pozice a vlastnosti objektu) → tvorba interakcí uživatele s objektem → přehrávání vytvořeného příběhu → export příběhu.

Z toho vychází následující návrh. Uživatelské rozhraní a jeho řídící systémy je rozdělené do šesti hlavních částí/panelů:

Horní lišta Panel představující horní lištu typickou pro desktopové aplikace s GUI[2]. Obsahuje sadu tlačítek, která plní základní a časté akce. Do horní lišty lze zabudovat i další grafické prvky, například text zobrazující aktuální načtený projekt.

Editor scén Bezprostředně nejdůležitější částí aplikace je panel zobrazující virtuální svět, který uživatel buduje. Virtuální svět se skládá ze scén, každá scéna obsahuje objekty. Objekty ve scéně tvoří hierarchickou strukturu. Tento panel zobrazuje náhled do aktuálně načtené scény a obsahuje prvky, které umožňují uživateli vytvářet svůj virtuální svět a pohybovat se v něm. Zároveň panel slouží pro přehrávání a testování vytvořeného příběhu. Návrh panelu ve formě skici lze vidět na obrázku 3.2d.

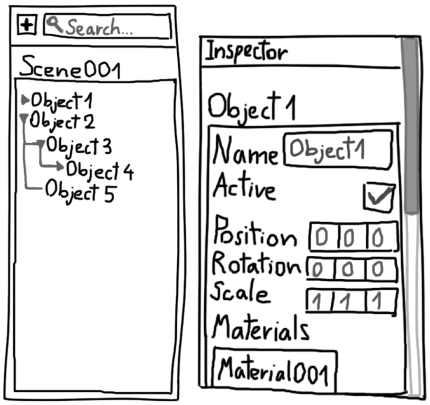
Editor interakcí Druhou nejdůležitější částí aplikace je rozhraní, které umožňuje uživateli vytvářet interakce, které promění statický svět v dynamický a umožňuje uživateli s ním interagovat. Skicu lze vidět na obrázku 3.2e.

Hierarchie Tento panel funguje v blízkém propojení s panelem editoru scén. Jak již bylo zmíněno, objekty ve scéně tvoří hierarchickou strukturu. Tento panel slouží pro kompaktní zobrazení všech objektů v dané scéně, včetně jejich hierarchických vztahů. Zároveň umožňuje tyto stavy měnit. Skicu lze vidět na obrázku 3.2a.

Zdroje Uživatel může do aplikace importovat vlastní 3D modely a obrazová data. Zároveň může vytvářet scény, ze kterých se skládá příběh. Tento panel slouží pro zobrazování importovaných a vytvořených objektů a umožňuje s nimi provádět související operace, například vkládat importované objekty do scény, nebo importované soubory opět mazat. Prvek v tomto rozhraní se nazývá tzv. asset. Skicu lze vidět na obrázku 3.2c.

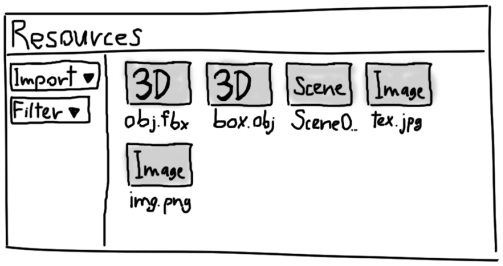
Inspektor Tato část představuje centrální místo, kde se zobrazuje většina dostupného nastavení pro aktuálně vybrané objekty. Vybraný objekt může být například asset, scéna nebo objekt v konkrétní scéně. Tento panel umožňuje vybrané objekty konfigurovat. To zahrnuje například změnu názvu nebo úpravu materiálů objektů ve scéně. Skicu lze vidět na obrázku 3.2b.

[2]Graphic User Interface – grafické uživatelské rozhraní.

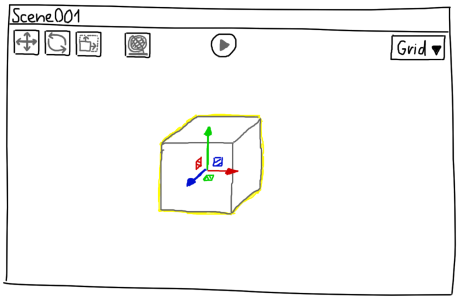


(a) : Skica panelu hierarchie.

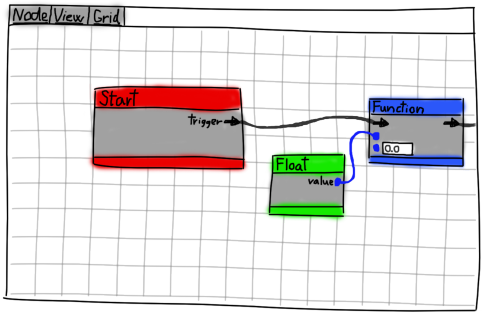
(b) : Skica panelu inspektor.



(c) : Skica panelu zdroje.



(d) : Skica panelu scény.



(e) : Skica grafu událostí.

Obrázek 3.2: Skici vybraných částí uživatelského rozhraní.

Další podpůrné řídící systémy a jejich případné grafické rozhraní jsou následující:

Správce vstupů Řídící systém, který spravuje uživatelské vstupy. Aplikace vyžaduje, aby klávesové zkratky měly význam závislý na stavu, ve kterém se aplikace nachází. Některé vstupy by měly být v určitých stavech kompletně neaktivní. Tento systém uplatňuje návrhový vzor `observer`. Jiné systémy mohou být pozorovatelé a když to aktuální stav aplikace dovoluje, systém vysílá události informující pozorovatele o tom, že uživatel zadal vstup. To umožňuje řešit například případy, kdy uživatel píše text do textového pole a je nežádoucí, aby tím zároveň spouštěl nesouvisející klávesové zkratky. Systém je snadno rozšiřitelný pro další uživatelské vstupy. Používá `Unity Input System`, což zároveň umožňuje snadné překonfigurování klávesových zkratek.

Kontextové menu Podpůrný systém pro dynamické budování kontextových menu v závislosti na současném kontextu (konkrétně v jaké části obrazovky uživatel kontextové menu vyvolal).

Plovoucí okna Plovoucí okna slouží pro zobrazení dalších informací a grafických prvků, které přímo nesouvisí s žádným z předešlých systémů,

například potvrzovací okna, když uživatel vykoná nebezpečnou akci.

Tooltip „Bublina“ s nápovědou, která se zobrazí po chvíli, co uživatel namíří kurzorem na určitý grafický element, která bude obsahovat stručný text vysvětlující význam elementu a případně související klávesovou zkratku.

Jednotlivé panely jsou v grafickém rozhraní oddělené a rozmístěné po okně aplikace. Zároveň by měly být responzivní v rámci okna aplikace a rozlišení monitoru.

3.5.1 Editor scén a hierarchie

Pro reprezentaci scény lze použít tzv. graf scény, což je n-ární strom. Objekty ve scéně mají právě jednoho předchůdce a mohou mít libovolný počet potomků. Za kořen stromu, tedy uzel, který nemá předchůdce, lze považovat scénu samotnou.

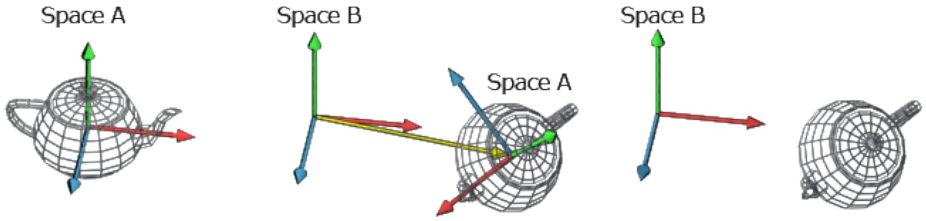
Výhodou této kompozice je možnost shlukovat množiny objektů do skupin. To umožňuje pohybovat s jednotlivými objekty ve scéně individuálně, ale zároveň vytvářet skupiny, které se pohybují současně. Jedním příkladem je například stůl, na kterém jsou postavené předměty. Uživatel může určit lokální pozici předmětů v závislosti na pozici stolu a zároveň přemisťovat stůl a všechny předměty se pohybují s ním. Druhým příkladem je model budovy složený z individuálních modelů zdí, oken, střechy, dveří atd. Opět je užitečné mít možnost hýbat s celou budovou současně.

Je to smysluplné také matematicky. 3D objekty jsou složené z vrcholů, které mají svoji geometrickou polohu v prostoru. Topologické informace určují, jaké vrcholy spolu tvoří hrany, plošky atd. Geometrická poloha jednotlivých vrcholů je vyjádřena v lokálním souřadnicovém prostoru objektu, tedy předpokládá, že střed objektu leží v počátku souřadnicového systému. Samotné objekty pak obsahují informaci o své poloze, rotaci a rozměru ve virtuálním světě, například v takzvané transformační matici. Tento koncept je vyobrazen na obrázku 3.3, kde všechny vrcholy čajníku jsou vyjádřeny v souřadnicovém prostoru A. Pozice čajníku ve virtuálním světě je určena transformační maticí, která definuje přechod mezi souřadnicovým prostorem B.

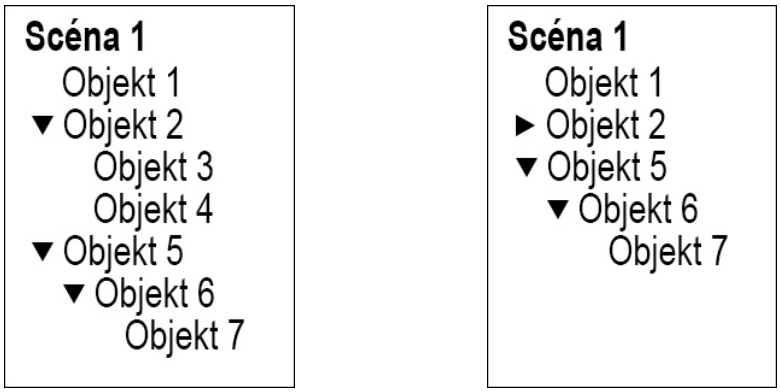
V případě grafu scény má každý objekt ve scéně uloženou svoji transformaci relativní ke svému přímému předchůdci. Pro zjištění světové polohy vrcholu objektu stačí znát všechny transformace v hierarchii od daného objektu po kořen stromu.

Hierarchie

Panel hierarchie zobrazuje hierarchii aktuálně načtené scény. Všechny objekty ve scéně mají své zástupné jméno, které uživatel může měnit. Jak napovídá název, hierarchie slouží k zobrazení a konfiguraci hierarchických vztahů objektů ve scéně. Uživatelské rozhraní má vzhled vertikálně řazeného seznamu objektů ve scéně. Potomci jsou v seznamu vždy pod svým předchůdcem. Jejich hloubka v hierarchickém stromě je vyznačena odsazením. Potomci jsou



Obrázek 3.3: Příklad rozdílných souřadnicových systémů. [27]



(a) : Objekt s rozevřenou hierarchií. **(b) :** Objekt se zavřenou hierarchií.

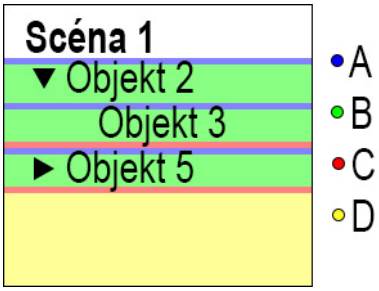
Obrázek 3.4: Příklad hierarchie scény.

v rozevíracím menu a lze je skrýt tlačítkem u jejich předchůdce. Vizuální návrh tohoto skrytí lze vidět na obrázcích 3.4a a 3.4b.

Uživatel může hierarchické vztahy objektů měnit přesouváním objektů v hierarchii na jiné pozice. Na obrázku 3.5 lze vidět čtyři rozdílné zóny, kam může uživatel přesunout objekt v hierarchii. Každý grafický element objektu má v hierarchii tři zóny: A, B a C.

Když uživatel přesune objekt do zóny A elementu, zařadí se nad tento element a budou mít stejného předchůdce. Pokud objekt přesune do zóny B objektu, stane se jeho posledním potomkem. Pokud ho přesune do zóny C objektu, zařadí se pod tento element a budou sdílet předchůdce. Speciální případ lze na obrázku 3.5 vidět u Objektu 2, který nemá červenou zónu C. Je to z toho důvodu, že v případě, že objekt má rozevřený seznam svých potomků, tato zóna by se chovala neintuitivně (v případě obrázku by zařadila objekt do stejné hierarchické úrovně jako je Objekt 2 nad Objekt 5, ale pod Objekt 3).

Poslední zónou je zóna D, která vyplňuje zbytek prostoru panelu hierarchie. Při přetažení objektu do této zóny se stane posledním objektem v hierarchii, v první úrovni (tedy nemá předchůdce / předchůdce je scéna samotná). Vhodné je přidat i grafické identifikátory toho, kam bude prvek zařazen pokud ho uživatel na dané pozici upustí.



Obrázek 3.5: Zóny pro změnu hierarchických vztahů.

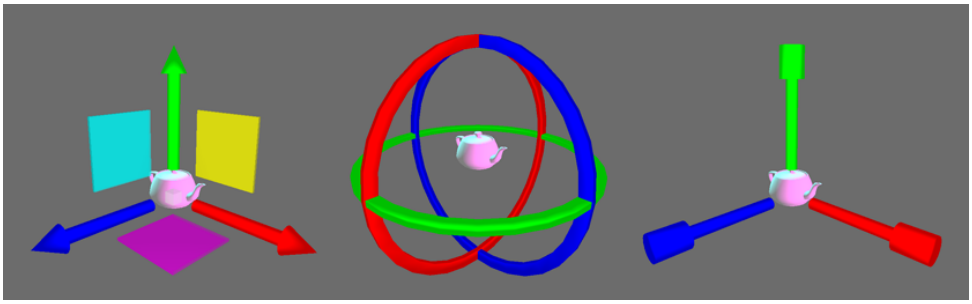
Editor scén

Základní stavební prvek virtuálního světa aplikace je takzvaný `scene object` (objekt scény). Vše ve virtuálním světě, co uživatel vidí a s čím může interagovat, je objekt scény. Objekt scény má ve scéně svoji pozici, rotaci a měřítko. Dále má typicky nějakou geometrii, která ho reprezentuje. Geometrie má na sobě jeden či více materiálů. Materiál určuje, jak světlo interaguje s povrchem geometrie, což určuje, jak objekt vypadá z pohledu uživatele. Na materiál mohou být naneseny i textury.

Objekty scény mohou mít speciální typ, který rozšiřuje jejich vlastnosti a funkce od statických modelů. Speciální typy jsou následující:

- Textový objekt, který reprezentuje text ve scéně. Uživatel může nastavit obsah, velikost, barvu a rozměry pomyslného rámečku, ve kterém je text umístěn.
- Obrazový objekt, který umožňuje vkládat do scény obrázky. Umožňuje nastavovat obsah a velikost/rozlišení.
- Světlo, které představuje bodový zdroj světla ve scéně. Lze měnit jeho barvu, intenzitu a dosah.
- Spínač, který reprezentuje místo/oblast ve scéně, ve které se má vykonat nějaký druh interakce. Tedy umožňuje detekovat, kdy do této oblasti uživatel vstoupil a kdy z ní vystoupil. Využívá se pro tvorbu interakcí. Vyskytuje se ve dvou tvarech – kvádr a koule.
- Hráč, který představuje postavu uživatele při spuštění příběhu. Lze nastavit, jestli má hráč kolidovat s geometrií ve scéně a jestli na něj má působit gravitace.

Panel, ve kterém se vykresluje scéna, se nazývá tzv. `scene view`. Grafické rozhraní se vykresluje ve 2D, tedy není důvod používat pro jeho zobrazování perspektivní kameru a stačí kamera ortografická. Avšak scéna již obsahuje 3D objekty a proto může být vhodné pro její zobrazení využívat perspektivní kameru. Aplikace tedy využívá více kamer. Hlavní ortografická kamera, která vykresluje celé uživatelské rozhraní a tzv. `scene view kamera` vykreslující



Obrázek 3.6: Gizmo manipulátory. [28]

scény, která je v základu perspektivní, ale měla by fungovat správně i v ortografickém módu, který se může při budování virtuálních světů hodit.

Editor scén umožňuje uživateli postavit si svůj virtuální svět. Do scény se uživatel dívá skrze editorskou kameru, kterou se může po světě při jeho budování pohybovat. Aby uživatel mohl s objekty ve scéně manipulovat, musí je nejdřív vybrat. To lze buď tím, že na ně klikne myší, nebo vybráním jeho grafického elementu v panelu hierarchie. Aby bylo zřejmé, jaký objekt je vybraný, zbarví se jeho obrys.

Pro manipulování s objekty ve virtuálním světě se využívají tři druhy manipulátorů, tzv. `gizmo` nástroje na změnu pozice (translace), změnu rotace a změnu měřítka/velikosti (scale). Systémy jako Unity, Blender, UE4, Maya všechny řeší vizuální vzhled a chování manipulátorů trošku rozdílně, ale všechny sdílí stejný základ. Ve všech zmíněných systémech má každá hlavní osa daného souřadného systému svoji specifickou barvu a manipulátory jsou těmito barvami obarveny. V některých zmíněných systémech je zároveň možné zobrazit kombinaci těchto manipulátorů současně (například Unity a Blender). Uživatel s manipulátory myší manipuluje a tím se aplikují změny na vybraný objekt.

Příklad manipulátorů lze vidět na obrázku 3.6. Když uživatel vybere objekt, vybraný manipulátor se objeví. Pozice manipulátoru ve scéně může být různá, intuitivní je umístit manipulátor na stejnou pozici jako je pozice objektu ve scéně. Manipulátory se vykreslují přes ostatní objekty, aby s nimi mohl uživatel interagovat a nebyly zakryté objekty ve scéně. Zároveň by měly být z pohledu kamery stále stejně veliké, aby s nimi šlo manipulovat nezávisle na vzdálenosti kamery od objektu a velikosti objektu.

Translace Manipulátor sloužící pro translaci (posouvání) objektů se skládá z několik částí. Prvním jsou prvky sloužící k posouvání objektu v jedné konkrétní ose, které mají tvar šipky. Doplňují je prvky pro posouvání objektu ve dvou osách, které jsou například na obrázku 3.6 vyobrazeny jako čtverce. Některé další systémy přidávají ještě další ovládací prvek, který může mít například formu krychle v počátku modelu manipulátorů, sloužící pro posuv objektu v rovině rovnoběžné k průmětně kamery. Za tyto prvky uživatel myší „tahá“ a tím posouvá vybraný objekt.

Rotace Pro rotaci se využívají kružnice nebo podobné útvary jako mezikruží

či anuloid. Rotací těchto těles po obvodu rotuje objekt v konkrétní ose. Opět lze doplnit ještě jedním kruhovitým objektem pro rotaci kolem osy určené směrem pohledu kamery.

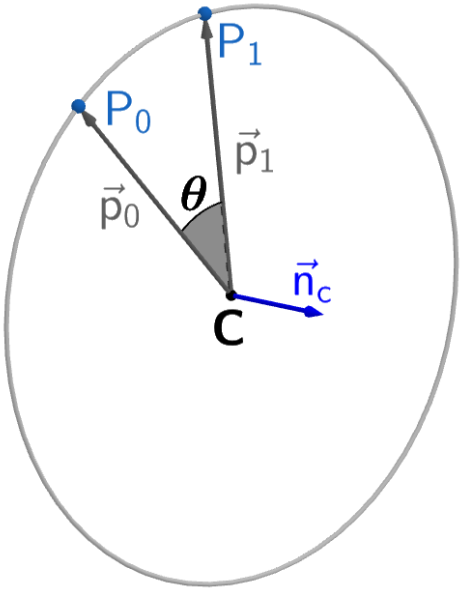
Měřítko Tento manipulátor umožňuje měnit rozměry/škálu objektu v jednotlivých osách. Vizuálně vypadají podobně jako šipky pro translaci, jen místo špičatých kuželů na konci jsou jiné geometrické útvary, například krychle nebo válce. Dá se opět doplnit ještě jeden ovládací prvek v počátku modelu manipulátorů, který mění škálu uniformně ve všech třech osách.

Implementačně se pro změnu pozice a škály v jedné ose využívá podobný výpočet. Úkolem je vyčíslit vzdálenost, kterou kurzor při tahání za ovládací prvky urazil ve směru dané osy. Výpočet nejbližšího bodu na přímce určené danou osou k přímce ve směru paprsku vrženého kamerou do pozice kurzoru ve scéně lze řešit analyticky. Tento výpočet se provádí každý snímek, kdy uživatel za ovládací prvek „tahá“. Rozdíl tohoto bodu v současný snímek a bodu z předchozího snímku dá vektor, o který je třeba v případě translace vybraný objekt posunout. V případě měřítka je situace o něco složitější, protože je nutné řešit, v jakém směru se má měřítko měnit. Pomocí znaménka z výsledku skalárního součinu mezi vektorem pohybu a směrem dané osy lze zjistit, jestli je směr pohybu po ose či proti ní. Velikost (magnitude) vektoru pohybu uplatněnou na tento směr je ještě třeba vynásobit dvěma, jelikož škálování objektu probíhá od jeho počátku, což je typicky ve středu objektu, tedy se zvětšuje na obě strany. Tento výpočet má ještě poslední nedostatek a to je speciální případ, kdy na počátku škálování objektu měl objekt v dané ose zápornou škálu. V takovém případě je třeba otočit směr škálování na opačnou stranu.

Pro změnu rotace se využívá podobný přístup, kdy je za úkol najít nejbližší bod na kružnici určené kruhovitým objektem k přímce určené paprskem vrženým z kamery do pozice kurzoru. Nejdříve se spočítá průsečík paprsku s rovinou, ve které leží kružnice. Následně se provede projekce tohoto průsečíku na obvod kružnice. Tento výpočet se opět provádí každý snímek kdy uživatel s tělesem otáčí. Z bodu na kružnici na současném snímku a bodu na kružnici na předchozím snímku lze vypočítat úhel otočení θ. Problém je v tom, že tento úhel je vždy pozitivní, tedy není zřejmé, jakým směrem uživatel těleso otáčí. Směr otáčení lze získat jako znaménko následujícím výpočtem (značení je vizualizováno na obrázku 3.7).

$$sign = (\vec{p}_0 \times \vec{p}_1) \cdot \vec{n}_c.$$

Transformovat objekt pomocí manipulátorů lze v různých souřadných systémech a pro budování světů je užitečné, když uživatel může mezi prostory přepínat. Základní jsou globální (světový) souřadný systém a lokální souřadný systém transformovaného objektu. Některé systémy nabízí i transformace v lokálním prostoru svého předchůdce v hierarchii. Výjimkou je manipulátor škály, u kterého není intuitivní výsledek škálování v jiném než v lokálním souřadném systému objektu, a proto se nepoužívá.



Obrázek 3.7: Výpočet úhlu otočení u manipulátoru rotace.

Pokud uživatel drží při selekci objektů ve scéně klávesu Ctrl, je možné přidávat více objektů do selekce, čemuž se říká tzv. multiselekce. Stejným způsobem lze vybírat více objektů současně v panelu hierarchie. Transformace pomocí manipulátorů se pak aplikují na všechny vybrané předměty. Zde se objevuje několik implementačních myšlenek. První je otázka, kde mají být umístěny manipulátory. Jedna možnost je spočítat průměr z pozic všech vybraných objektů a manipulátory umístit tam. To může být matoucí, protože pak manipulátor může být i na místě, kde se nevyskytuje ve scéně žádná geometrie. Proto je v tomto případě přirozenější používat pozici i lokální prostor objektu, který byl přidán do selekce jako poslední.

Při implementaci je třeba myslet ještě na jednu věc, a to je fakt, že hierarchické vztahy způsobují, že není možné při transformaci více objektů zároveň jednoduše vzít každý objekt a uplatnit na něj transformaci, jelikož potomci jsou v souřadných systémech svých předchůdců, tedy i kopírují transformace svých předchůdců. To by mohlo způsobit, že by byla transformace na potomka uplatněna vícekrát. Jednoduchý příklad je případ, kdy uživatel vybral dva objekty, A a B. Objekt B je potomkem objektu A. Uživatel pomocí manipulátorů posune vybrané objekty o jednu jednotku vzdálenosti po určité ose. Jelikož je ale objekt B potomkem objektu A, posune se o jednu jednotku ve svém lokálním prostoru a o další jednotku jelikož je v prostoru svého předchůdce, tedy se posune o dvě jednotky vzdálenosti. Pro hlubší hierarchické struktury by se transformace aplikovaly tolikrát, kolik má vybraný objekt po cestě do kořene vybraných předchůdců. Proto je třeba při aplikování transformací vybrané objekty filtrovat a transformace aplikovat jen na nejvyšší vybrané zástupce v daných podstromech hierarchie.

Pro flexibilnější použití manipulátorů nabízí různé systémy řadu dalších zlepšení. Jedním z nich je možnost provádět transformace po uživatelem

nastavených krocích. V případě translace to znamená, že se objekty pohybují po středech buněk neviditelné uniformní mřížky, kde buňky mají velikost nastavenou uživatelem. V případě rotace si uživatel nastaví krok v jednotkách úhlů.

3.5.2 Editor interakcí

Tvorba interakcí je způsob, jak udělat z jinak statického virtuálního světa něco, co s uživatelem komunikuje a s čím může interagovat. Flexibilní prostředí pro vytváření rozmanitých interakcí vyžaduje nějakou formu programování či skriptování. Vzhledem k tomu, že aplikace má za cílovou skupinu i uživatele, kteří nemusí umět programovat, hodí se pro vytváření interakcí využití vizuálního skriptování, které bylo popsáno v sekci 2.2.

Pro tento projekt byla vybraná forma uzlového vizuálního skriptování. Ačkoli výzkum uvedený v [4] naznačuje, že blokové skriptování je o něco málo intuitivnější, je třeba brát v potaz, že testy v této práci byly prováděné na krátkých scénářích, reprezentujících krátkou část skriptu/funkce. Jak již bylo zmíněno, základní problém vizuálního skriptování je malá kompaktnost kvůli velikosti grafických elementů reprezentujících bloky či uzly, což vede v horší škálovatelnost pro větší projekty oproti textovému skriptování. U blokového skriptování je tento problém dále umocněn faktem, že bloky je třeba napojovat na sebe ve specifických pozicích a tvoří souvislý blok, čili uživatel má o to menší flexibilitu v uspořádání grafických prvků na obrazovce. I z pohledu náročnosti implementace grafického rozhraní je jednodušší vytvořit jednoduché „krabičky“ vizualizující uzly spojené křivkami, než vytvářet flexibilní bloky variabilních tvarů, které musí dynamicky měnit svou velikost v závislosti na napojování dalších bloků.

Řídící systém se nazývá **graf událostí** (event graph). Graficky je reprezentován jako 2D plocha, na kterou uživatel umisťuje uzly, přezdívané „krabičky“. Krabička reprezentuje část kódu v textovém skriptování, tedy to může být například celá funkce, nebo pouze datová položka. Jednotlivé krabičky lze spojovat křivkami, takzvanými „spoji“. Spoje jsou dvojího typu. První typ jsou exekuční/kontrolní spoje, které reprezentují pořadí, ve kterém se krabičky vyhodnocují (spouštějí). V kontextu textového skriptování tedy určují pořadí vyhodnocování jednotlivých částí kódu. Druhý typ jsou datové spoje, které slouží k předávání dat mezi jednotlivými krabičkami. V textových programovacích jazycích jsou základním stavebním kamenem funkce. Typická funkce může mít na vstupu nějaká vstupní data, která zpracuje a na výstup pošle výstupní data formou návratové hodnoty. Tato vstupní a výstupní data jsou v grafu událostí předávaná pomocí datových spojů. Zároveň v programovacích jazycích se musí funkce spustit (zavolat) jinou částí kódu a během svého vykonávání může volat další funkce. Tato volání jsou v grafu událostí reprezentována pomocí exekučních spojů.

Krabičky se spojují pomocí spojů v místech nazývaných porty. Port má tedy celkem čtyři druhy: datový vstupní, datový výstupní, exekuční vstupní a exekuční výstupní. Krabička může mít libovolný počet libovolných typů portů. Je poměrně vzácné, že krabičky mají více než jeden vstupní exekuční

port, ale není to nemožné. Funkcionalita krabičky se pak odvíjí od toho, z jakého exekučního vstupu přišel spouštěcí signál. Na exekuční vstup může být napojeno více spojů současně. Naopak z exekučního výstupu může vést pouze jediný spoj, aby bylo deterministicky určeno pořadí vykonávání. Datové spoje fungují opačně. Do datového vstupu může vést pouze jediný spoj, aby byla deterministicky určena předaná hodnota. Z datového výstupu může vést více spojů, tedy návratovou hodnotu krabičky může přijímat na vstupu více jiných krabiček. Na spoje jsou kladené další omezující podmínky. Nelze vést spoj z výstupu krabičky do vstupu stejné krabičky. Datové spoje lze vytvářet pouze mezi porty stejných datových typů.

Mohou existovat i krabičky, které nemají žádné exekuční vstupy. Z tohoto pohledu lze krabičky rozdělit do tří hlavních následujících kategorií:

1. **Spustitelné (executable)**

   Do této kategorie spadají krabičky, které mají alespoň jeden exekuční vstup. V moment, kdy do této krabičky přijde exekuční signál, krabička se vyhodnotí a případně uloží výstupní hodnoty do datových výstupů. Nikdy jindy se nevyhodnocuje.

2. **Nespustitelné (non-executable)**

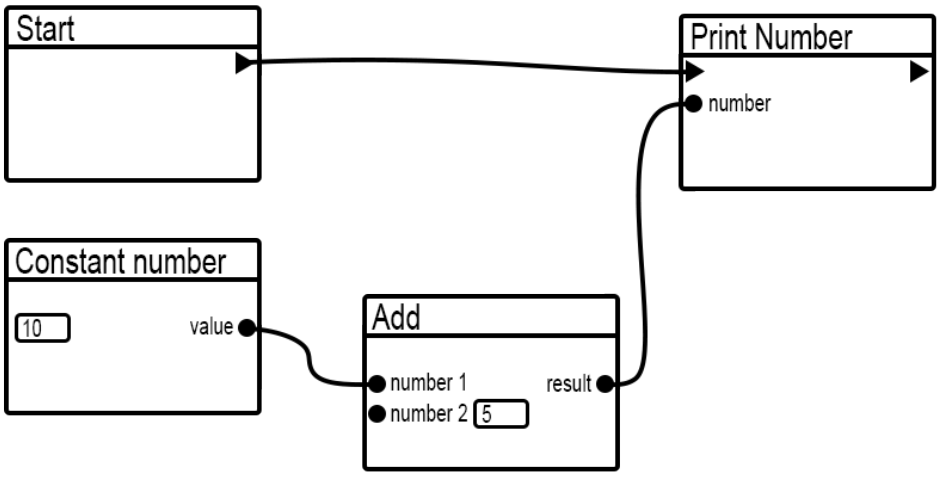
   V této kategorii jsou krabičky, které nemají žádný exekuční vstup ani exekuční výstup. K jejich vyhodnocení dochází pokaždé, když jiná krabička vyžádá hodnotu svého datového vstupu, který má spoj na datový výstup této krabičky. Tedy signál pro vyhodnocení přichází z datových výstupů krabičky.

3. **Události**

   Krabičky v této kategorii nemají exekuční vstup, ale mají exekuční výstup. Slouží jako generátory, které v grafu událostí vytváří exekuční signály. Reprezentují události, ke kterým ve virtuálním světě došlo. Mohou být závislé na uživatelském vstupu, například událost, která se vyvolá, když uživatel stiskne klávesu. Další možností jsou události závislé na stavu scény, například událost vygenerovaná při prvotním spuštění příběhu, nebo událost, která se vyvolá ve chvíli, kdy dojde ke kolizi dvou objektů ve virtuálním světě.

Ve chvíli, kdy má být krabička vyhodnocena, porty datových vstupů vyžádají své hodnoty. Pokud má port datových vstupů spoj, vyžádá tuto hodnotu od napojené krabičky. Pokud je tato napojená krabička nespustitelného typu, touto žádostí se vyhodnotí. Pokud je napojená krabička spustitelného typu, poskytne hodnotu uloženou z jejího posledního vyhodnocení. Pokud spoj nemá, použije se výchozí hodnota. U specifických datových typů může uživatel tuto výchozí hodnotu specifikovat přímo v krabičce. Například v případě, že se jedná o číselnou hodnotu, krabička obsahuje textové pole, do kterého může výchozí hodnotu zadat.

Pro obrázek 3.8 by po spuštění příběhu vyhodnocování probíhalo následovně:



Obrázek 3.8: Typy uzlů v grafu událostí.

1. Spuštění příběhu způsobí, že krabička `Start` vytvoří exekuční signál, který vyšle do krabičky `Print Number`.
2. Krabička `Print Number` dostane exekuční signál. Pro své vyhodnocení vyžádá hodnotu vstupního datového portu `number`.
3. Jelikož do datového portu `number` vede spoj, nepoužije se výchozí hodnota, ale vyžádá se od krabičky, ze které spoj vede, tedy krabičky `Add`.
4. Krabička `Add` je nespustitelná, tedy se v této chvíli vyhodnotí.
5. Pro vyhodnocení vyžádá hodnoty vstupních datových portů `number 1` a `number 2`.
6. Jelikož do datového portu `number 1` vede spoj, vyžádá se hodnota od krabičky, ze které spoj vede, tedy krabičky `Constant number`. Do té nevedou žádné spoje a vrátí zapsanou statickou hodnotu 10.
7. Jelikož do datového portu `number 2` nevede spoj, použije se výchozí hodnota v textovém poli, tedy hodnota 5.
8. Krabička `Add` získá všechny potřebné vstupní hodnoty pro své vyhodnocení, tedy se může vyhodnotit. Spočítá součet hodnot a výstupním datovým portem pošle výsledek.
9. Krabička `Print Number` získá všechny potřebné vstupní hodnoty a vyhodnotí se. Vzhledem k tomu, že z exekučního výstupu nevede spoj, exekuční signál zanikne.

Krabičky mohou dále obsahovat další prvky, které slouží ke specifikaci toho, jak se má krabička vyhodnotit. Typickým příkladem mohou být krabičky provádějící matematické operace. Místo toho, aby bylo třeba vytvářet jednotlivé krabičky pro každou matematickou operaci mezi dvěma čísly, jako je součet,

násobení, dělení atd., lze vytvořit rozevírací menu, které obsahuje nabídku všech smysluplných operací.

3.5.3 Zdroje

Panel zdrojů slouží k zobrazení dostupných zdrojů, pomocí kterých uživatel může stavět virtuální světy. To mohou být importované 3D objekty, obrazová data, vytvořené scény a jakékoli další zdroje, pro které může být v budoucnu vytvořená podpora.

Každý zdroj má v aplikaci svůj grafický prvek, který obsahuje jeho název a ikonu. Pro importovaná obrazová data lze jako ikonu použít jejich miniaturu. Jelikož zdrojů může být ve větších projektech veliké množství, lze prvky filtrovat v závislosti na jejich typu. Užitečná je i možnost vyhledávání ve zdrojích na základě jejich názvu. Panel zdrojů umožňuje jednotlivé zdroje vybírat pomocí myši pro zobrazení jejich nastavení. Umožňuje také zdroje mazat z projektu.

Unity a další herní enginy používají pro jakýkoliv obsah, který lze používat v herních světech termín „herní asset“. Asset může reprezentovat 3D modely, textury, mapy, zvuky, textové soubory apod. Ve StoryGraph lze rozlišovat dva základní druhy assetů: objekt scény, takzvaný „scene object asset“, a obrázek, takzvaný „texture asset“.

Scene object asset představuje jakousi šablonu objektu, ze které lze vytvářet kopie/instance a vkládat je do scén. Pokud je možné tuto šablonu upravovat, tento přístup představuje značnou výhodu při budování virtuálních světů. Místo toho, aby uživatel vytvářel instance objektu ve scéně a pokaždé jim znovu musel nastavovat parametry materiálu, textury a další, nastaví tyto parametry jednou přímo na šabloně objektu a nově vytvořené instance už budou mít tyto parametry nastavené jako výchozí.

3.5.4 Inspektor

Aplikace má za cíl, aby byla přehledná a jednoduchá na ovládání. Inspektor panel slouží jako centrální místo pro zobrazování informací a nastavení, aby je uživatel nemusel hledat na více místech v aplikaci. Ostatní řídící systémy inspektoru předávají informace o tom, jaká data a nastavení má zobrazit. Nastavení se zobrazuje v oddělených komponentách. Pokud uživatel vybere objekt ve scéně, zobrazí se základní nastavení objektu, například jeho název a pozice ve scéně. Dále pokud objekt obsahuje animace, současně se zobrazí animační komponenta, obsahující seznam dostupných animací. Dále pokud se jedná o specifický typ objektu, například světlo, zobrazí se komponenta světla, která umožňuje nastavit parametry jako sílu, směr, dosah, barvu světla apod. Tento systém komponent zaručuje jednoduchou rozšiřitelnost aplikace pro další typy objektů a konfigurovatelných nastavení.

3.5.5 Horní lišta

Pro počítačové aplikace s grafickým rozhraním je běžné, že obsahují horní lištu, která obsahuje rozevírací položky s tlačítky. Pro horní lištu je typické, že je viditelná v jakékoli části rozhraní, tedy že ji nepřekrývají jiné grafické prvky. Hodí se proto pro funkce, které by uživatel mohl chtít uživatel vyvolat v jakémkoli stavu aplikace. Obsahuje například funkcionality pro vytvoření nového projektu, ukládání projektu, načítání uložených projektů, ukončení aplikace a další. Velmi častá je záložka „Help“, která typicky obsahuje užitečné odkazy, návody, dokumentaci, kontakt pro nahlašování chyb a další.

3.5.6 Kontextové menu

Další velice užitečný systém pro aplikaci je systém pro dynamické vytváření kontextových menu. Kontextová menu představují flexibilní systém, jak uživateli představit nabídku možností, které v daném kontextu může vykonat přímo na místě, na které se soustředí, například na pozici kurzoru. Kontextové menu může obsahovat rozevírací prvky, které obsahují další kontextové menu, což umožňuje vložit do kontextových menu obrovské množství možností. Pokud lze kontextové menu vyvolat na libovolné pozici v aplikaci, je vhodné zaručit, že nabídka bude vždy viditelná a vejde se na obrazovku.

3.5.7 Plovoucí okna

Lze narazit na situace, kdy je třeba v aplikace zobrazit situační grafické prvky, se kterými uživatel běžně nepracuje a není vhodné, aby byly v rozhraní stále zobrazené. Příkladem může být globální nastavení, u kterého se předpokládá, že si ho uživatel nastaví jednou a nebude mít potřebu ho měnit příliš často. Další příklad jsou situace, kdy je třeba uživateli zobrazit zprávu, u které je důležité, aby ji uživatel nepřehlédl, například chybové hlášky. Další příklad jsou potvrzovací okna, které se zobrazí v případě, že uživatel provede potenciálně nebezpečnou akci, například se pokusí aplikaci ukončit aniž by uložil změny v projektu. Pro tyto situace se hodí grafické prvky zobrazit ve formě plovoucího okna, které se zobrazí v místě, kde si ho uživatel všimne a po vyžádané interakci jednoduše zavře.

3.6 Export

Aby bylo projekty možné exportovat a zpětně importovat, musí být navržen formát ve kterém bude projekt uložen. Pro tuto aplikaci se hodí textový formát, který je lidsky čitelný. To umožňuje, aby uživatelé mohli projekt měnit i mimo prostředí aplikace. Zároveň je jedním z požadavků zadání projektu, aby byl formát navržený tak, aby jej bylo použít v jiných vizualizačních nástrojích. Inspirací pro tento formát mohou být formáty VRML a X3D, popsané v sekci 2.5, které umožňují ukládání celých herních světů. Tyto formáty avšak neřeší popis struktur, jako je graf událostí.

Jelikož je aplikace vyvíjená v jazyce C#, je smysluplné použít pro proces serializace existující nástroje či framework, který tuto práci usnadní. Unity nabízí pro serializaci do formátu JSON třídu `JsonUtility`, ale tato třída není příliš rozšířená a umožňuje jen velice základní serializaci. Vhodnější volba je framework `Json.NET`, který umožňuje serializovat a deserializovat libovolné `.NET` objekty.

Není nutné ani žádoucí exportovat všechny data v projektu. Aby byly výsledné soubory co nejjednodušší, nejpřehlednější a nejkratší, měly by obsahovat jen nezbytné informace pro zpětnou deserializaci. Proto je vybrána podmnožinu objektů, které je nutné serializovat a pro každý z nich je navržené rozhraní, které obsahuje pouze důležité informace. Aby bylo možné exportované projekty použít i v dalších vizualizačních nástrojích, nesmí se formát spoléhat na žádná interní data a procesy aplikace StoryGraph. Proto může být nezbytné do serializovaného rozhraní přidat i informace, bez kterých by se při načítání aplikace StoryGraph obešla, ale jen díky tomu, že zná vnitřní strukturu všech objektů před serializací.

Serializovat je nutné všechny všechny importované assety. Dále je nutné serializovat všechny scény a jejich obsah, což znamená veškeré objekty ve scéně a pro každou scénu její graf událostí. V grafu událostí je třeba serializovat všechny uzly a všechny spoje. U uzlů je třeba myslet na to, že mohou obsahovat uživatelem nastavené výchozí hodnoty na portech datových vstupů, které je nutné také serializovat.

Druhou část exportu tvoří balíček importovaných souborů, které je nezbytné exportovat spolu se serializovanými daty, aby bylo možné projekt načíst na jakémkoli počítači. Jelikož StoryGraph neukládá importované objekty do žádných vlastních vnitřních formátů, stačí importované soubory zkopírovat do projektové složky. Vylepšit tento přístup by šlo tím, že by byly soubory převedeny do komprimované podoby, které mohou značně zmenšit objem dat a tím pádem velikost výsledného projektu.

Kapitola 4

Implementace

V této kapitole bude popsán proces a detaily implementace. Budou popsány použité vývojářské nástroje, co při implementaci činilo největší problémy a jaké byly v implementaci využité specifické přístupy pro tvorbu grafického rozhraní. Kapitola bude také obsahovat snímky obrazovky z hotové aplikace. Aplikace byla implementovaná v souladu s návrhem popsaným v kapitole 3, ale nebyly implementované všechny popsané funkcionality.

4.1 Vývojářské nástroje a externí zdroje

Aplikace byla vytvořena v herním engine Unity verze 2020.3.33f1. Veškerý zdrojový kód byl napsán jako Unity skripty v jazyce C#. Pro psaní bylo využito IDE[1] Visual Studio Community 2019. Visual Studio má tu výhodu, že lze přímo propojit s Unity a využívat tak při hledání chyb debugovací funkce jako Breakpointy (body přerušení). Projekt byl verzován pomocí webového repozitáře GitLab. Modely gizmo manipulátorů byly vytvořeny v grafickém editoru Blender.

Pro vývoj byla využitá řada balíčků z oficiálního `Unity Asset Store` obchodu a veřejných repozitářů. V následujícím textu budou externí zdroje vyjmenovány a stručně popsáno jejich využití v aplikaci. Odkazy na tyto zdroje jsou uvedeny v příloze B.

- **Unity UI Extensions** je balíček grafických prvků, rozšiřující standardní nabídku Unity. V projektu se využivá upravený skript `UILineRenderer`, pomocí kterého lze vykreslovat Bézierovy křivky určením čtyř řídících bodů. Využívá se pro vykreslování spojů mezi krabičkami v grafu událostí. Dále se v aplikaci využívá grafický prvek `Color Picker`, pomocí kterého lze vybírat barvu materiálů, světel a textu z interaktivní palety.
- **Dynamic Panels** obsahuje assety, pomocí kterých lze vytvořit dynamické panely. Panely lze přemisťovat, měnit jejich velikost a zachycovat je v různých místech obrazovky. Lze je vkládat přes sebe a přepínat mezi nimi. Tento asset je využitý pro základní vzhled uživatelského rozhraní, který si uživatelé mohou nastavit dle své potřeby.

[1] Integrated Development Environment – vývojové prostředí.

- **Quick Outline** je balíček obsahující skript, který lze vložit jako komponentu na objekt a pomocí shaderů a materiálů v balíčku vytvoří obrys objektu. Tento skript se využívá pro zvýraznění vybraných objektů při budování scény.

- **TriLib 2** poskytuje velice důležitou funkcionalitu. Když uživatel importuje vlastní soubory s 3D objekty do Unity, Unity engine si tyto soubory interně zpracovává a vytváří si pro ně vlastní modifikace a ukládá je ve vlastním proprietárním formátu. Importované modely mají k dispozici řadu nastavení, jak má Unity soubor interně zpracovat. To zahrnuje například otázky jako jestli má být pro model vygenerována kolizní geometrie, jestli mají být pro 3D geometrii modelu prováděny optimalizace, jestli se mají importovat či vygenerovat normály modelu, jestli se mají importovat materiály, textury, animace apod. Problém je v tom, že tento import nelze provádět za běhu aplikace, což ve StoryGraph znemožňuje uživateli importovat si vlastní 3D modely bez implementace vlastního loaderu 3D objektů. TriLib umožňuje importovat řadu populárních formátů pro 3D modely za běhu aplikace. Tento balíček obsahuje další užitečné balíčky jako je **Unity Standalone File Browser**, který slouží jako wrapper pro dialogová okna pro procházení a výběr souborů na několika operačních systémech, což je další věc kterou Unity za běhu aplikace nepodporuje, ale aplikace to vyžaduje. Nevýhodou tohoto browseru je to, že neumožňuje vybírat více souborů současně a neumožňuje otevírat a ukládat celé složky, což komplikuje proces ukládání a načítání projektů.

- **Unity Converters for Newtonsoft.Json** umožňuje serializovat a deserializovat speciální třídy pro datové typy které využívá Unity, jako jsou `Vector3`, `Quaternion`, `Color` a další. Serializovat tyto typy je jinak problematické. Například datový typ `Vector3` obsahuje cyklické reference, protože každý vektor obsahuje odkaz na svůj normalizovaný tvar.

- **Icons8** je webová stránka s širokou databází ikon, které umožňuje v omezeném rozlišení stáhnout zdarma. Většina ikon v aplikaci byla získána z této stránky.

4.2 Import souborů

Aplikace umožňuje importovat 3D modely ve formátech fbx, obj a stl. Importování modelů je řešené pomocí balíčku TriLib 2, zmíněného v sekci 4.1. Dále aplikace umožňuje import obrazových dat ve formátech jpg a png, což jsou formáty, které umí Unity načítat i za běhu.

Při importování souborů se soubory současně zkopírují do speciálního adresáře, které Unity vytváří pro každý projekt. Lokalita této složky je na každém OS[2] jiná a adresa je uložená v řetězci `Application.persistentDataPath`. Toto kopírování je nutné, protože kdyby uživatel po importování souboru do

[2]Operační Systém.

aplikace změnil jeho lokalitu v systému, aplikace by ztratila přehled o tom, kde se nachází a nebylo by možné ho exportovat s projektem.

Pro každý importovaný objekt, ať už obrázek nebo 3D model, se vytvoří v aplikaci objekt, který dědí ze třídy `BaseAsset`. Jelikož importované soubory lze z aplikace opět mazat, musí si každý asset udržovat reference na všechny další objekty v aplikaci, které tento asset používají. Při smazání souboru se tyto reference rozpojí. Všechny objekty využívající obrázek jako texturu musí ze sebe texturu smazat. Všechny instance objektů využívající geometrii z importovaných 3D objektů se smažou ze scény.

4.3 Scéna

Scéna, kterou uživatel vidí přes panel `scene view`, je renderovaná pomocí odlišné kamery, než přes kterou se uživatel dívá na uživatelské rozhraní – **kamera scény**. Tato kamera nevykresluje na frame buffer, ale do speciálního typu textury, takzvané `Render Texture`. Tato textura je nanesena na panel scény a tím uživatel vidí scénu. Tento přístup představuje problém, pokud chce uživatel interagovat s objekty na této textuře. Pro detekci, jestli uživatel kliknul na daný objekt, vyšle Unity paprsek z kamery skrz pozici myši do scény a zjistí, jestli paprsek protnul kolizní geometrii objektu. Jelikož ale objekt na dané pozici ve skutečnosti není, protože se uživatel dívá pouze na vykreslenou texturu, bylo třeba vytvořit vlastní upravený `Raycaster`, který pozici myši přepočítá na lokální pozici uvnitř panelu scény a paprsek vyšle přepočítanou pozicí.

Pomocí grafu událostí lze při přehrávání příběhu upravovat scénu, objekty ve scéně se mohou skrývat, měnit svoji pozici atd. Nebylo by ale žádoucí, že by tyto změny byly uchované i po zastavení příběhu. Proto když uživatel spustí příběh, vytvoří se **kopie** aktivní scény a příběh se spustí v této kopii. Po zastavení příběhu se tato kopie smaže a zobrazí se zpátky neupravená původní scéna.

4.4 Export

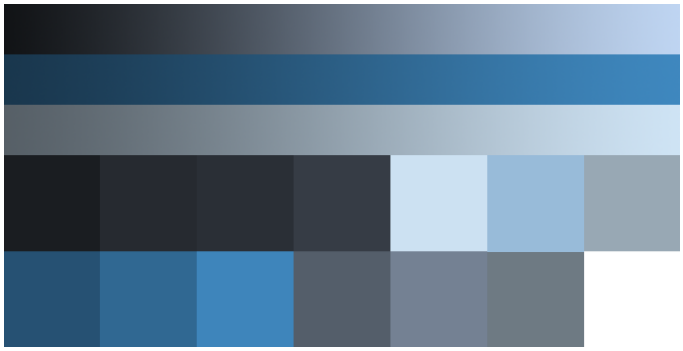
Všechny třídy, které reprezentují objekty, které je třeba ukládat/exportovat, mají druhou, datovou třídu, která obsahuje veřejné pole pro všechny datové položky, které je třeba exportovat. Serializovatelné objekty obsahují veřejnou metodu `Serialize`, která vytvoří instanci datové třídy a naplní veřejné pole daty. Naplnění daty probíhá uvnitř konstruktoru datových tříd. Kód 4.1 obsahuje příklad datové třídy scény.

4.5 Uživatelské rozhraní

V první fázi byla zvolena barevná paleta, které by se aplikace měla držet. Bylo vybráno několik zástupných barev, které využívá většina grafických

```
[Serializable]
public class SceneData
{
    public string ID;
    public string Name;
    public EventGraphData EventGraph;
    public List<object> SceneObjects;
    public SceneData() { }
    public SceneData( Scene src )
    {
        this.ID = src.ID;
        this.Name = src.Name;
        this.EventGraph = src.Graph.Serialize();
        this.SceneObjects = new List<object>();
        foreach ( var o in src.SceneObjects.Values )
        {
            this.SceneObjects.Add( o.Serialize() );
        }
    }
}
```

Kód 4.1: Příklad datové třídy scény.



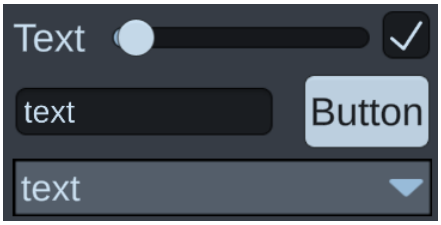
Obrázek 4.1: Barevná paleta aplikace.

prvků v aplikaci. Zvolené barevné přechody a zástupné barvy lze vidět na obrázku 4.1.

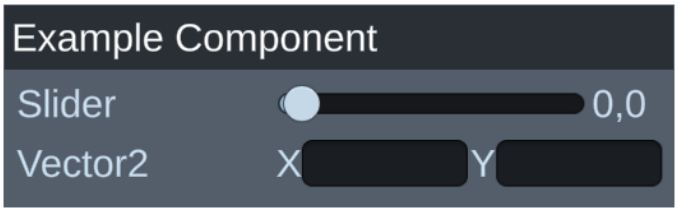
Poté byly vytvořeny základní grafické prvky, například stylizovaný text, checkbox, dropdown, slider atd. Díky `Prefab` systému Unity lze tyto grafické prvky pak využívat napříč celou aplikací. Kdyby se třeba později v implementaci zjistilo, že je třeba grafické prvky nějak upravit, stačilo by upravit konkrétní `prefab` a tato změna by se propagovala do všech instancí v grafickém rozhraní. Část základních grafických prvků lze vidět na obrázku 4.2.

Podobným způsobem byly vytvořeny stavební bloky pro grafické prvky v konkrétních panelech, například inspektor a obsah uzlů v grafu událostí. Pro tyto stavební bloky byly zároveň vytvořeny skripty, které spravují jejich vzhled a funkcionalitu. Skripty obsahují veřejné rozhraní, pomocí kterého mohou nadřazené komponenty obsahující tyto stavební bloky zjišťovat, jestli uživatel změnil hodnotu – například napsal text do textového pole.

Stavební bloky lze skládat a tím vytvářet složitější grafické prvky v aplikaci. Praktický příklad lze vidět na obrázku 4.3, kde je grafický element komponenty v panelu inspektoru. Na obrázku 4.4 lze vidět jeho grafickou strukturu,



Obrázek 4.2: Základní grafické prvky v aplikaci.



Obrázek 4.3: Příklad komponenty v inspektoru.

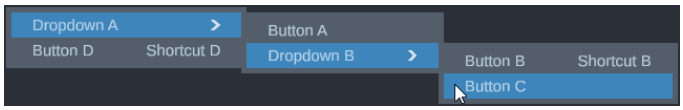
```
Komponenta
├─ Hlavička
│  └─ UI_Text "Example Component"
└─ Obsah
   ├─ Řádek
   │  ├─ UI_Text "Slider"
   │  └─ UI_SliderInput
   └─ Řádek
      ├─ UI_Text "Vector2"
      └─ UI_Vector2
         ├─ UI_Text "X"
         ├─ UI_FloatInput
         ├─ UI_Text "Y"
         └─ UI_FloatInput
```

Obrázek 4.4: Grafická struktura komponenty na obrázku 4.3.

složenou ze stavebních bloků.

V aplikaci se velmi často využívá možnost dynamického budování kontextových menu v závislosti na místě, kde uživatel v aplikaci klikne pravým tlačítkem. Kontextové menu se skládá podobně jako jiné části uživatelského rozhraní ze stavebních bloků. Může obsahovat řadu tlačítek a rozevíracích menu. Každé rozevírací menu pak může obsahovat další tlačítka nebo další rozevírací menu. O stavbu kontextových menu se v kódu stará třída `ContextMenuBuilder`, pomocí které lze kdekoli v kódu pomocí několika řádků kódu vytvořit libovolné kontextové menu. Příklad kontextového menu lze vidět na obrázku 4.5. V kódu 4.2 lze vidět jak bylo toto menu vytvořeno.

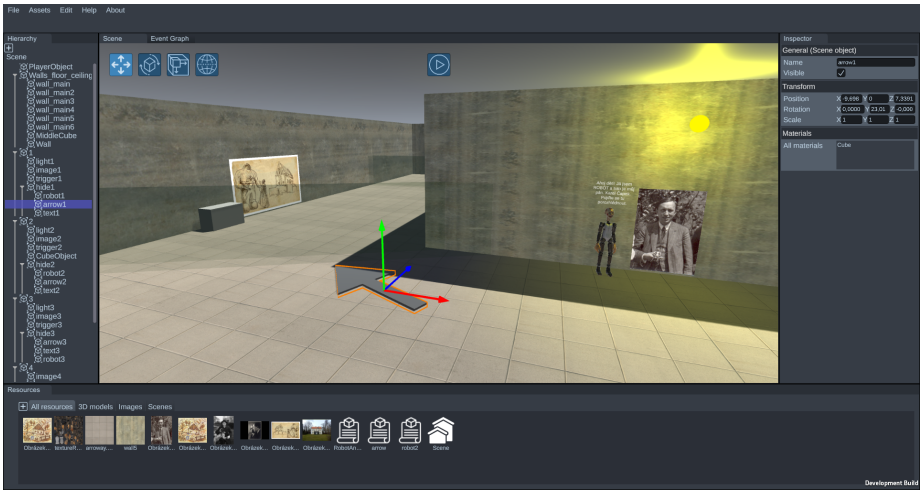
Na obrázcích 4.6 a 4.7 lze vidět snímky obrazovky z aplikace.



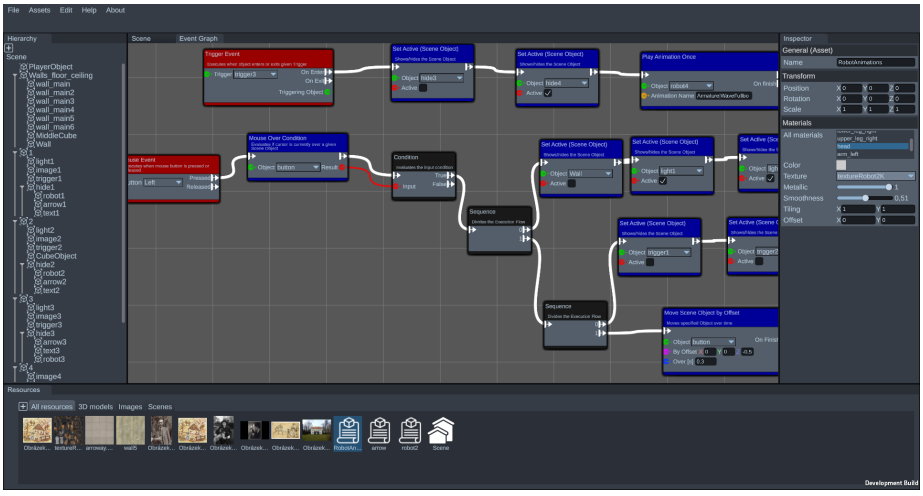
Obrázek 4.5: Příklad kontextového menu v aplikaci.

```
ContextMenuBuilder.Instance.BuildContextMenu(
    Dropdown( "Dropdown A",
        Button( () => MethodToCall(), "Button A" ),
        Dropdown( "Dropdown B",
            Button( () => MethodToCall(), "Button B", "Shortcut B" ),
            Button( () => MethodToCall(), "Button C" )
        )
    ),
    Button( () => MethodToCall(), "Button D", "Shortcut D" )
);
```

Kód 4.2: Příklad části kódu pro vytvoření kontextového menu na obrázku 4.5.



Obrázek 4.6: Snímek obrazovky se záložkou scény.



Obrázek 4.7: Snímek obrazovky se záložkou grafu událostí.

4.6 Srovnání

Tato podkapitola obsahuje srovnání vytvořené aplikace StoryGraph se systémy, které byly analyzovány v sekci 2.4.3. Tabulka 4.1 obsahuje zhodnocení aplikace StoryGraph a má stejnou strukturu jako tabulka 2.1, ve které byly zhodnoceny analyzované existující systémy.

Největší slabinou aplikace StoryGraph je nedostatečná bezpečnost a to zejména absence systému *Undo* a *Redo* na pohyb o krok zpět a vpřed. Jejich implementace byla v plánu, ale z časových důvodů se odložila v zájmu implementace jiných důležitých částí aplikace. Budoucí rozvoj aplikace je popsán v sekci 4.9.

Implementace tvorby interakcí pomocí textových skriptovacích jazyků není současně v plánu. Důvodem je, že aplikace má primárně cílit na skupinu uživatelů, kteří neumí programovat a textový skriptovací jazyk by nevyužili. Zároveň by implementace byla složitá a mohla by trvat dlouho, zatímco v aplikaci chybí mnoho dalších důležitých funkcí.

Funkce	StoryGraph
Import obrazových dat	5/5
• běžné formáty (jpg, png)	✔
• alfa kanál	✔
Import 3D modelů	5/5
• běžné formáty (obj, stl, fbx)	✔
• podpora animací	✔
Vytváření světů	5/5
• neomezená plocha	✔
• členění do scén	✔
• vertikální dimenze	✔
Tvorba interakcí	3/5
• vizuální skriptování	✔
• textové skriptování	✘
Export	3/5
• spustitelné v editoru	✔
• standalone aplikace	✘
Efektivita	4/5
• klávesové zkratky	✔
• přizpůsobitelné rozhraní	✔
Jednoduchost	4/5
• přehledné/čisté rozhraní	✔
• tooltipy, nápovědy	✘
• průvodce	✔
Bezpečnost	2/5
• automatické ukládání	✘
• krok zpět a vpřed	✘
• potvrzovací okna	✔
Celkem	**3,9/5**

Tabulka 4.1: Funkcionality aplikace StoryGraph.

4.7 Tvorba nových uzlů

Graf událostí byl navržen způsobem, aby bylo co nejjednodušší vytvářet nové typy uzlů s vlastní funkcionalitou. V této sekci bude krátký průvodce strukturou uzlu ve zdrojovém kódu a návod, jak vytvořit nový libovolný uzel. Nejprve je nutné vytvořit nový C# skript obsahujícího třídu definující uzel. Každý uzel musí dědit ze třídy `Node`, na což je potřeba v hlavičce souboru použít `using EventGraph`, viz kód 4.3:

```
1  using EventGraph;
2
3  public class MyNode : Node {
4      // ...
5  }
```

Kód 4.3: Deklarace třídy uzlu.

Následně lze ve třídě definovat vstupy, výstupy a rozevírací menu s možnostmi, viz kód 4.4:

```
1  using EventGraph;
2
3  public enum Options { A, B, C };
4
5  public class MyNode : Node {
6      FlowInPort inFlow; // flow input port
7      FlowOutPort outFlow; // flow output port
8      ValueInPort<float> input; // input value port (needs to
           specify datatype)
9      ValueOutPort<bool> output; // output value port (needs to
           specify datatype)
10     EnumDropdown<Options> options; // dropdown with enum values
11     // ...
12 }
```

Kód 4.4: Deklarace vstupů a výstupů uzlu.

Pomocí modifikátoru `override` lze rozšířit inicializační metodu. Uvnitř této metody lze nastavit vizuální vzhled uzlu, jeho barvu a text portů a dalších grafických elementů, viz kód 4.5:

```
1  // ...
2  public override void Init( NodeGraph graph )
3  {
4      base.Init( graph );
5      Graphics.Title = "My Node"; // change node title
6      Graphics.Subtitle = "Description"; // change node subtitle
7      Graphics.BackGroundColor = Color.green; // change node color
8      inFlow.Label = "Start"; // change port label
9      Graphics.ResizeToMinSize(); // update node's size
10 }
11 // ...
```

Kód 4.5: Příklad inicializační metody uzlu.

Pomocí modifikátoru `override` se implementuje metoda `OnExecute`, která definuje, co se má stát, když má být uzel vyhodnocen, viz 4.6:

```
    // ...
    public override void OnExecute()
    {
        base.OnExecute();
        var option = options.CurrentSelection; // gets value of
            dropdown
        var portValue = input.GetValue(); // gets value of data input
            port
        output.Value = true; // sets value of data output port
        outFlow.Execute(); // sends execution signal through flow
            output port
    }
    // ...
```

Kód 4.6: Příklad exekuční metody uzlu.

Existují situace, kdy může být žádoucí, aby uzel vykonal určitou akci v okamžiku, kdy je příběh v dané scéně spuštěn nebo ukončen. To lze pomocí modifikátoru `override` na metody `OnPlayStart` resp. `OnPlayStop`, viz 4.7:

```
    // ...
    public override void OnPlayStart()
    {
        base.OnPlayStart();
        // ...
    }
    public override void OnPlayStop()
    {
        base.OnPlayStop();
        // ...
    }
    // ...
```

Kód 4.7: Metody uzlu volané při spuštění a vypnutí příběhu.

Uvnitř uzlu lze vytvořit a používat jakékoli další pomocné pole a metody.

Někdy může být žádané, aby se funkcionalita, kterou uzel spouští, vykonávala po určitou dobu, například aby se objekt ve scéně posunul plynule během časového úseku a ne okamžitě, skokovou změnou na novou pozici. Pro tyto účely se perfektně hodí využít koprogramy (`Coroutine`), které Unity podporuje.

Unity koprogram vypadá uvnitř třídy jako běžná metoda s návratovou hodnotou `IEnumerator`, ale lze ji v určitých místech přerušit a pokračovat v ní až na dalším snímku, čehož je docíleno pomocí příkazu `yield return null`.

Pro spuštění koprogramů je třeba použít příkaz `StartCoroutine`. Pro přerušení koprogramu se používá příkaz `StopCoroutine`. Koprogramy mohou běžet na jakémkoli objektu, který dědí z třídy `MonoBehaviour`. Překážkou je to, že koprogramy mohou běžet pouze na aktivním objektu ve scéně. V případě StoryGraph, pokud není otevřený panel grafu událostí, všechny uzly jsou neaktivní a tím pádem na nich nelze spouštět koprogramy za běhu příběhu.

Proto se ve StoryGraph pro spouštění koprogramů využívá třída `NodeCoroutines`, která je na objektu, který je ve scéně vždy aktivní, a obsahuje statické metody pro spouštění a přerušení koprogramů, které běží na libovolných uzlech. Použití koprogramů lze vidět v kódu 4.8.

```
1     public override void OnExecute()
2     {
3         base.OnExecute();
4         NodeCoroutines.Start( DoSomething( 10f ) );
5     }
6     IEnumerator DoSomething( float val )
7     {
8         // ...
9     }
```

Kód 4.8: Příklad spouštění koprogramu uvnitř uzlu.

4.8 Dokumentace

Horní lišta aplikace obsahuje tlačítko, které otevře pdf soubor obsahující uživatelskou příručku. Příručka je dostupná v příloze C. Příručka obsahuje následující části:

- Základní popis rozhraní aplikace, funkce jednotlivých panelů a k čemu slouží.
- Popis scény, funkce tlačítek v panelu scény, návod k ovládání kamery a pohybu ve scéně.
- Popis hierarchie, návod k vytváření hierarchických vztahů mezi objekty.
- Popis grafu událostí, návod k pohybu, vytváření nových krabiček a spojů mezi nimi. Klávesové zkratky v grafu událostí. Popis vnitřní struktury uzlů, rozdělení uzlů do tří hlavních kategorií a popis jednotlivých kategorií.
- Souhrnný popis typů uzlů, které aplikace aktuálně obsahuje. Příručka neobsahuje vysvětlení všech typů uzlů, ale obsahuje dostatek informací pro obecné pochopení funkcionality grafu událostí.
- **Popis formátu exportu, vysvětlení struktury výsledného datového souboru a jednotlivých jeho položek.**

4.9 Plánovaná rozšíření

Kvůli nedostatku času nebylo realistické implementovat vše, co bylo popsané v návrhu. Zde je seznam věcí, které nebyly implementovány, ale jsou zájmem budoucího vývoje a při návrhu se s nimi počítalo:

- **Undo/Redo**, neboli zvrátit poslední vykonanou akci uživatele, případně ji po zvrácení opět vykonat. Vzhledem k tomu, že aplikace obsahuje

funkční systém ukládání a načítání projektů a většina důležitých dat je tím pádem serializovatelná, neměla by implementace být složitá.

- **Rozšíření gizmo manipulátorů**. Aplikace současně obsahuje pouze manipulátory pro pohyb v jedné ose. Nabídku by bylo vhodné rozšířit alespoň o manipulátory pro pohyb ve dvou osách pro translaci a rotaci a manipulátor pro uniformní škálování. Součástí tohoto rozšíření by byla i možnost nastavení pevných kroků při pohybu manipulátorů.
- **Vyhledávání v seznamech**. V současné chvíli veškeré seznamy, například seznam objektů v hierarchii, seznam materiálů objektu a seznamy objektů ve specifických uzlech v grafu událostí nemají možnost v těchto seznamech vyhledávat. Proto pokud jsou tyto seznamy dlouhé, je zdlouhavé a neefektivní v nich vyhledávat. Proto by bylo vhodné přidat textová pole, které umožní v těchto seznamech filtrovat a vyhledávat.
- **Import hdr skybox souborů**, jelikož v současné chvíli je v projektu pouze jeden výchozí skybox, uživatelé nemají možnost si nastavit, jak vypadá okolní svět. Součástí načítání těchto souborů by bylo i nastavení intenzity světla, které prostředí vyzařuje.
- **Širší nabídka speciálních objektů**. Nabídka speciálních objektů by šla rozšířit v mnoha směrech. V současnosti lze například vytvářet ve scénách pouze bodová světla, nabídku by tedy bylo možné rozšířit o reflektory a směrová světla. Další užitečný objekt by byly kamery, které by uživatel mohl umisťovat ve scénách a přepínat mezi nimi.
- **Tooltips**, neboli grafické elementy ve formě malých oken s textem, které se objeví při namíření ukazatele na specifické UI prvky. Okna by obsahovala nápovědu, či vysvětlení funkce prvku.
- **Více typů shaderů**. V současnosti aplikace pro vykreslování modelů ve scéně využívá jeden obecný shader, což znamená, že aplikace nepodporuje speciální typy materiálů, například poloprůhledné materiály.

Kapitola 5
Uživatelské testování

Uživatelské testování použitelnosti proběhlo na vytvořené aplikaci. Tato kapitola popisuje metodiku testování, průběh provedeného testování a nakonec jsou výsledky testování vyhodnoceny a na základě nalezených problémů je navrženo několik změn v aplikaci.

5.1 Metodika testování

Dle [29] je vhodné testování použitelnosti provádět vícekrát s malým počtem účastníků v každém kole (autor uvádí jako ideální počet tři). Testovaní účastníci totiž nemusí nutně narazit na všechny problémy, ale pouze nejdůležitější problémy v každé iteraci. Tři účastníci velmi pravděpodobně na tyto problémy narazí. Problémy se odstraní a poté může proběhnout další iterace testování. Vhodné je testovat například každý měsíc vývoje.

Autor dále uvádí, že v začátku vývoje není příliš důležité, jací účastníci budou pro testování vybráni. Hledání specifických lidí z cílové skupiny uživatelů typicky vyžaduje více práce a na začátku vývoje je pravděpodobné, že návrh bude obsahovat závažné problémy pro jakékoliv uživatele. Tedy není nutné hledat konkrétní cílové uživatele z úzké skupiny, naopak není vhodné provádět testování pouze na cílové skupině uživatelů. Pokud testovaný systém vyžaduje znalost nějaké domény, je nutné v testování zahrnout nějaké účastníky s touto znalostí, ale nemusí to být všichni.

Testování použitelnosti závisí na pozorování jednoho účastníka, který se pokouší systém používat pro vyřešení typických úkolů. Testování může probíhat v řízeném prostředí, například v laboratoři použitelnosti. Úkoly mohou vycházet z průzkumu cílů uživatelů, pro pokrytí testovaných částí, nebo například k ověření hypotézy. Účastníci testu jsou vybráni podle účelu testu (často podle persony). [30]

5.2 Průběh testování

Testování se zúčastnili 4 účastníci. Účastníkům bylo nejprve vysvětleno, k čemu má aplikace sloužit. Dále byl popsán způsob, jak bude testování probíhat. Testování trvalo s každým účastníkem v rozmezí 40-70 minut. Účastníci

měli buď velice malé, nebo žádné znalosti programovacích jazyků. Předchozí znalosti účastníků jsou shrnuty v tabulce 5.1.

	Programování	Grafika
Účastník A	Webové programování	Cinema 4D, AutoCAD, SolidWorks
Účastník B	Žádné	Žádné
Účastník C	Žádné	Cinema 4D
Účastník D	Žádné	Žádné

Tabulka 5.1: Předchozí znalosti testovaných účastníků.

5.2.1 Scénář

Účastníci si stáhli balíček, který obsahoval spustitelnou aplikaci StoryGraph a složku, ve které byl jeden soubor s 3D modelem a jeden soubor s obrazovými daty.

- **Úloha 1** Spusťte aplikaci StoryGraph a seznamte se s uživatelským rozhraním.
- **Úloha 2** Vložte do úvodní scény podlahu a tu rozšiřte. Dále vložte do scény kostku a posuňte ji ve scéně tak, aby ležela na podlaze.
- **Úloha 3** Zduplikujte kostku a kopii posuňte tak, aby ležela na první kostce. Nastavte hierarchický vztah tak, aby spodní kostka byla předchůdcem horní kostky.
- **Úloha 4** Přejmenujte objekty kostek. Spodní kostka by měla mít název „kostka1“ a horní kostka název „kostka2“. Dále nastavte, aby na hráče v příběhu působila gravitace. Nakonec nastavte, aby horní kostka měla modrou barvu.
- **Úloha 5** Přepněte se do panelu grafu událostí. Vytvořte následující interakci se scénou: Po spuštění příběhu se kostka1 posune o 2 jednotky nahoru v ose Y za dobu tří sekund. Vytvořte další interakci: když uživatel během přehrávání příběhu stiskne mezerník, přehrávání příběhu se ukončí. Interakce vyzkoušejte ve spuštěném příběhu. Nakonec vymažte libovolný vytvořený spoj mezi vytvořenými uzly.
- **Úloha 6** Importujte do projektu 3D model v přiložené složce. Importujte do projektu obrazová data v přiložené složce. Importovaný objekt vložte do scény a nastavte texturu objektu na importovaný obrázek. Pokuste

se upravit parametry materiálu modelu tak, aby vypadal co nejvíc realisticky.

- **Úloha 7** Projekt uložte a ukončete aplikaci.

Úkoly byly navrženy tak, aby je bylo možné splnit rychle, ale aby zároveň pokryly veškeré hlavní funkcionality aplikace, tedy import souborů, stavění a konfigurace virtuálního světa, vytváření interakcí a export.

Po splnění úkolů probíhala rozprava nad tím, co činilo největší problémy, jaká byla účastníkova očekávání a jak se lišila od skutečnosti. Účastníkům byla položená otázka, co by v aplikaci změnili a co v aplikaci postrádali.

5.2.2 Testování

Účastník A U úlohy 3 účastník nenašel, jakým způsobem vytvořit hierarchické vztahy mezi objekty. Na tomto úkolu se zasekl, po vysvětlení funkce hierarchie se mu nakonec úlohu podařilo vyřešit. V úloze 5 použil účastník špatný typ uzlu, která objekt posune na specifickou světovou pozici, místo uzlu, který objekt posune ze současné pozice o zadanou odchylku. Zbytek úloh účastník splnil bez problémů.

Účastník B Při prozkoumávání rozhraní v úloze 1 účastník vytvořil novou scénu a nevěděl, jak se přepnout zpět do té původní. Jednalo se o nepochopení konceptu dělení projektu na scény, ne o problém uživatelského rozhraní. Účastník si otevřel uživatelskou příručku a seznámil se tak s ovládáním. Chvilku trvalo, než našel způsob, jak do scény přidávat objekty. U úlohy 3 ani tento účastník nevymyslel, jak nastavit hierarchický vztah mezi kostkami. U úlohy 5 se pokoušel účastník přetáhnout objekty z hierarchie do grafu událostí. Po chvíli přišel na to, jak přidávat nové uzly, ale nepřišel na to, co jsou to porty a že lze uzly s jejich pomocí spojovat. Uzly posunul tak, aby byly v grafu za sebou a tím značily posloupnost vykonání. Ani po krátkém vysvětlení, co to jsou porty, účastník z vysvětlení nepochopil, jaké porty lze spolu spojovat a jak se graf vyhodnocuje. U úlohy 6 se účastník pokoušel aplikovat texturu na 3D objekt přetažením prvku assetu z panelu zdrojů na objekt ve scéně.

Účastník C U úlohy 3 se objevil stejný problém jako u obou předchozích účastníků, účastník nepochopil, jak vytvořit hierarchický vztah mezi kostkami. U úlohy 4 při nastavování barvy chvilku trvalo, než účastník přišel na to, že při kliknutí v panelu inspektor na materiál v seznamu se rozbalí další část menu, kde může nastavit barvu kostky. U úlohy 5 účastník nevymyslel rozdíl mezi datovými a exekučními porty a jak funguje exekuční tok. Po stručném vysvětlení úlohu splnil bez dalších problémů. U úlohy 6 se i tento účastník pokoušel přetahovat ikony z panelu zdrojů do scény.

Účastník D I u tohoto účastníka se objevil problém při nastavování hierarchických vztahů v úloze 3. Po zdůraznění pojmu hierarchie však po

chvilce úlohu splnil. U úlohy 5 účastníka zaujaly uzly pro spouštění animací. Z výkladu zadání úlohy si vyložil, že posunutí kostky je forma animace. Po chvilce se účastníkovi povedlo najít správné uzly a úlohu splnit. U úlohy 6 chvilku trvalo, než účastník vymyslel, jak importovaný objekt vložit do scény. Také se pokoušel přetáhnout ikonu z panelu zdrojů přímo do scény.

5.3 Výsledky testování

Úspěšnost účastníků v plnění jednotlivých úloh ve scénáři je shrnuta v tabulce 5.2. Zelená značí úspěch, žlutá značí částečné problémy nebo delší doba řešení a červená značí neúspěch.

	U1	U2	U3	U4	U5	U6	U7
Účastník A	✔	✔	✘	✔	~	✔	✔
Účastník B	✔	✔	✘	✔	✘	~	✔
Účastník C	✔	✔	✘	~	~	~	✔
Účastník D	✔	✔	~	✔	~	~	✔

Tabulka 5.2: Úspěšnost účastníků při plnění testovacích úloh.

5.3.1 Návrh úprav

Zde jsou popsány navrhované změny v aplikaci před další iterací uživatelského testování. Některé změny byly navrženy od samotných uživatelů po testování a některé byly vyvozeny z vypozorovaných problémů při testování.

- Z tabulky 5.2 vyplývá, že největší problém činila úloha 3. Účastníci nebyli schopni přijít na to, že grafické elementy představující objekty v panelu hierarchie lze přetahovat na jiné pozice v hierarchii a tím vytvářet hierarchické vztahy. Všichni účastníci se pokusili objekty v hierarchii vybrat a hledali tuto možnost v kontextovém menu. Navrhovaná změna je vložení tlačítka do kontextového menu. Uživatel vybere selekcí objekty a pravým tlačítkem otevře kontextové menu s možností „Set As Parent“ na objektu v hierarchii, který se má stát jejich předchůdcem.
- Druhý největší problém působila úloha 5, která byla založená na principiálním nepochopení toho, jak funguje graf událostí. Ačkoli polovina účastníků využila při plnění úkolů uživatelskou příručku, bohužel v době, kdy testování probíhalo, byla uživatelská příručka neúplná a stručná a neobsahovala vysvětlení, jak graf událostí funguje. Po testování byla uživatelská příručka rozšířena.

- Dále měli účastnici u úlohy 5 potíže s nalezením správného typu uzlu pro plnění konkrétní úlohy. Někdy i otevřeli správné rozevírací menu a přes krabičku přejeli. Pomoci by mohl vyhledávací systém na základě klíčových slov. Zájmem může být i důkladnější průzkum toho, jak uživatelé chápou názvy uzlů v kontextovém menu.
- Pohyb po scéně. Jeden účastník postrádal funkcionalitu přibližování a oddalování ze současné pozice. Druhému účastníkovi vadilo, že pro navigaci po scéně musí držet pravé tlačítko myši. Přibližování vycházelo z účastníkovy předchozí zkušenosti v programu Cinema 4D a pro současné ovládání kamery se nehodí. Pro navigaci po scéně bez nutnosti stisku pravého tlačítka účastník A navrhl, aby uživatelské rozhraní obsahovalo tlačítko, pomocí kterého by šlo přepínat mezi současným způsobem ovládání a mezi stavem, kdy pravé tlačítko není třeba pro navigaci držet.
- Při rozpravě s účastníky po testování převažoval názor, že pro mazání spojů mezi uzly v grafu událostí by se mělo zobrazit kontextové menu při pravém kliknutí na spoj. V současné chvíli se při pravém kliknutí spoj rovnou smaže.
- Všichni účastníci se v úloze 6 pokoušeli přetahovat prvky z panelu zdrojů přímo do scény. Současně se polovina účastníků pokoušela do projektu soubory importovat přetažením přímo do okna aplikace. Tyto funkcionality by bylo vhodné přidat.
- Když je vybrán objekt, který obsahuje materiály, není žádný z těchto materiálů v panelu inspektor ve výchozím vztahu vybraný a rozevírací menu obsahující vlastnosti materiálu je tím pádem skryté. Toto by bylo vhodné změnit tak, aby ve výchozím stavu byl nějaký materiál vybraný a uživatelé na první pohled viděli grafické elementy pro změny parametrů materiálů.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a implementovat editor pro vytváření interaktivních 3D příběhů. Před návrhem proběhla analýza existujících systémů umožňujících tvorbu příběhů a principů tvorby uživatelského rozhraní.

Na základě analytické části byla vytvořená aplikace StoryGraph, pomocí které lze vytvářet jednoduché interaktivní příběhu. Do aplikace lze importovat vlastní 3D objekty, 2D obrazová data a textové anotace. Příběh je členěn na oddělené scény. Interakce s virtuálním světem lze tvořit pomocí vytvořeného blokového skriptovacího jazyka. Aplikace prošla uživatelským testováním použitelnosti, na jehož základě byly navrženy úpravy a rozšíření.

Aplikace splňuje zadané cíle práce, ale stále vyžaduje značné rozšíření funkcionalit, aby byla lépe použitelná v praxi. V porovnání s již existujícími systémy představuje aplikace střední cestu. Nemá tolik omezení pro vytváření interaktivních příběhů, jako jiné zkoumané systémy. Zároveň je méně komplexní, přehlednější a jednodušší pro pochopení a naučení, než rozšířené herní enginy. Aplikace je dobře rozšiřitelná a umožňuje pokračovat v dalším vývoji.

Bibliografie

1. MILLER, Carolyn Handler. *Digital Storytelling: A creator's guide to interactive entertainment*. 4th ed. Boca Raton: CRC Press, 2020. ISBN 978-1138341586.
2. *Scripting Vs. Programming: Difference between Scripting and Programming* [online]. JavaTpoint [cit. 2022-03-16]. Dostupné z: `https://www.javatpoint.com/scripting-vs-programming`.
3. *Programování vs. skriptování* [online]. cs.education-wiki [cit. 2022-03-16]. Dostupné z: `https://cs.education-wiki.com/5436584-programming-vs-scripting`.
4. UGGLA, Alexander. *Improving the User Experience of Visual Scripting Languages*. Norrköping, Sweden, 2021. Dostupné také z: `http://www.diva-portal.org/smash/record.jsf?dswid=7044&pid=diva2%3A1621764`. Dipl. pr. Linköping University, Department of Science, Technology, Media a Information Technology. Vedoucí práce Niklas RÖNNBERG.
5. HANSSON, Theresia. *How to Implement a Node-based Scripting Language: Part 1 - Game Pipeline* [online]. 2017-10-08 [cit. 2022-03-16]. Dostupné z: `http://gamepipeline.com/how-to-implement-a-node-based-scripting-language-part-1/`.
6. *Sandbox (computer security)* [online]. Wikimedia Foundation, 2022-03-10 [cit. 2022-03-16]. Dostupné z: `https://en.wikipedia.org/wiki/Sandbox_(computer_security)`.
7. *How works visual scripting ?* [Online]. FloPress [cit. 2022-03-16]. Dostupné z: `https://flopress.io/resource/visual-scripting/how-works-visual-scripting/`.
8. *About Unity Visual scripting* [online]. Unity Technologies, 2021-10-13 [cit. 2022-03-16]. Dostupné z: `https://learn.unity.com/tutorial/about-unity-visual-scripting?uv=2021.1&projectId=604a0d68edbc2a080cd78d78#604a1ebdedbc2a03f83ce9c8`.
9. MOLICH, Rolf; NIELSEN, Jakob. Improving a Human-Computer Dialogue. *Commun. ACM*. Břez. 1990, roč. 33, č. 3, s. 338–348. ISSN 0001-0782. Dostupné z DOI: `10.1145/77481.77486`.

10. NIELSEN, Jakob; MOLICH, Rolf. Heuristic Evaluation of User Interfaces. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Seattle, Washington, USA: Association for Computing Machinery, 1990, s. 249–256. CHI '90. ISBN 0201509326. Dostupné z DOI: 10.1145/97243.97281.

11. NIELSEN, Jakob. *10 Usability Heuristics for User Interface Design* [online]. Nielsen Norman Group, 2020-11-15 [cit. 2022-03-16]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/.

12. *CoSpaces Edu for kid-friendly 3D creation and coding* [online]. Delightex [cit. 2022-03-17]. Dostupné z: https://cospaces.io/edu/.

13. *CoSpaces Edu coding education* [online]. Delightex [cit. 2022-03-17]. Dostupné z: https://cospaces.io/edu/coding.html.

14. *What Is Serialization?* [Online]. Hazelcast [cit. 2022-05-19]. Dostupné z: https://hazelcast.com/glossary/serialization/.

15. *What is Serialization? & Serialization Types* [online]. W3computing [cit. 2022-05-19]. Dostupné z: https://www.w3computing.com/vb2008/understanding-serialization-types/.

16. *Data Serialization* [online]. Devopedia [cit. 2022-05-19]. Dostupné z: https://devopedia.org/data-serialization.

17. *BSON* [online]. Wikipedia [cit. 2022-05-19]. Dostupné z: https://en.wikipedia.org/wiki/BSON.

18. *JSON Example* [online]. JSON [cit. 2022-05-19]. Dostupné z: https://json.org/example.html.

19. ŽÁRA, Jiří. *Laskavý průvodce virtuálními světy* [online]. Brno: Computer Press, 1999 [cit. 2022-05-19]. Dostupné z: https://dcgi.fel.cvut.cz/LaskavyPruvodce/.

20. *Introduction to VRML* [online]. Advisory Group on Computer Graphics [cit. 2022-05-19]. Dostupné z: http://www.agocg.ac.uk/brief/vrml.htm.

21. *The Virtual Reality Modeling Language* [online]. Web3D Consortium [cit. 2022-05-19]. Dostupné z: https://www.web3d.org/documents/specifications/14772/V2.0/.

22. TIŠNOVSKÝ, Pavel. *Základy tvorby souborů typu X3D* [online]. Root.cz, 2008-01-31 [cit. 2022-05-19]. Dostupné z: https://www.root.cz/clanky/zaklady-tvorby-souboru-typu-x3d/.

23. *What is X3D?* [Online]. Web3D Consortium [cit. 2022-05-19]. Dostupné z: https://www.web3d.org/x3d/what-x3d.

24. *Extensible 3D (X3D) encodings: Part 1: XML encoding* [online]. Web3D Consortium [cit. 2022-05-19]. Dostupné z: https://www.web3d.org/documents/specifications/19776-1/V3.3/.

25. XIE, Jingming. Research on key technologies base Unity3D game engine. In: *2012 7th International Conference on Computer Science Education (ICCSE)*. 2012, s. 695–699. Dostupné z DOI: 10.1109/ICCSE.2012.6295169.

26. *Návrhové vzory GoF* [online]. ITnetwork.cz [cit. 2022-05-20]. Dostupné z: https://www.itnetwork.cz/navrh/navrhove-vzory/gof.

27. *World, View and Projection Transformation Matrices* [online]. Coding Labs [cit. 2022-05-19]. Dostupné z: http://www.codinglabs.net/article_world_view_projection_matrix.aspx.

28. DIAKOPOULOS, Dimitri. *tinygizmo* [https://github.com/ddiakopoulos/tinygizmo]. GitHub, 2017.

29. KRUG, Steve. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. 3rd ed. New Riders, 2014. ISBN 978-0321965516.

30. SCHMIDT, Jan. *Vyhodnocení: Testování Použitelnosti*. BI-TUR Tvorba uživatelského rozhraní [Přednáška]. ČVUT FIT, 2018.

Příloha A

Seznam použitých zkratek

2D Two-dimensional

3D Three-dimensional

AR Augmented reality

GUI Graphical user interface

HCI Human-computer interaction

HTML Hypertext Markup Language

IDE Integrated Development Environment

ISO International Organization for Standardization

JASS Just Another Scripting Syntax

JSON JavaScript Object Notation

MB Megabyte

NPC Non-Playable Character

OS Operační systém

RTS Real-time strategy

UE4 Unreal Engine 4

UI User interface

VRML Virtual Reality Modeling Language

X3D Extensible 3D

XML Extensible Markup Language

Příloha B

Zdroje třetích stran

- Unity UI Extensions – `https://github.com/Unity-UI-Extensions/com.unity.uiextensions`
- Dynamic Panels – `https://assetstore.unity.com/packages/tools/gui/dynamic-panels-114126`
- Quick Outline – `https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488`
- TriLib – `https://assetstore.unity.com/packages/tools/modeling/trilib-2-model-loading-package-157548`
- Unity Converters for Newtonsoft.Json – `https://github.com/jilleJr/Newtonsoft.Json-for-Unity.Converters`
- Icons8 – `https://icons8.com/`

Příloha C

Uživatelská příručka

StoryGraph User Guide

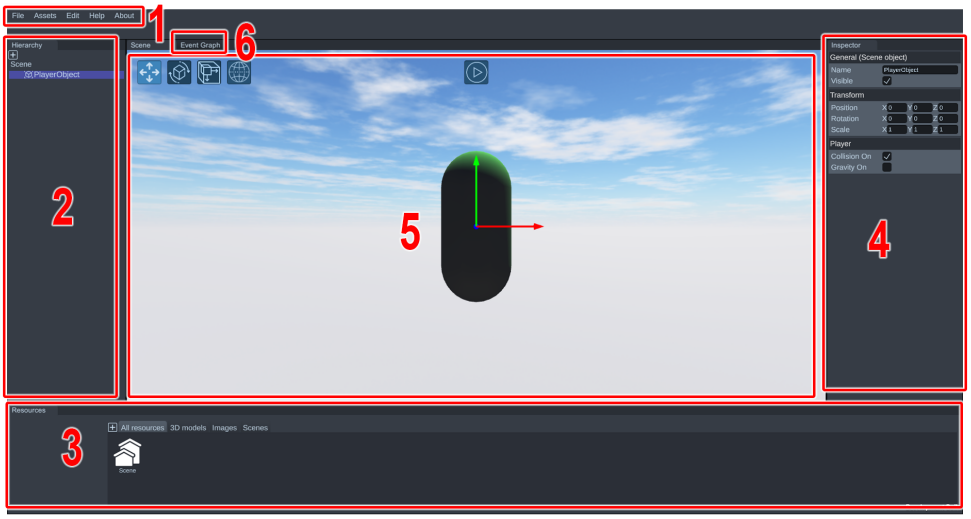
1. GETTING STARTED

StoryGraph is an application designed to create interactive virtual 3D worlds. Users can import 3D models and images, which they can use to build scenes. The key system for creating interactions with the world is called **Event Graph**. It utilizes a form of Visual Scripting, which uses Nodes and Connections to build interactions without previous knowledge of programming languages.

2. LAYOUT

The layout consists of 6 main panels.

1. **Top menu**, which contains general tools for project management.
2. **Hierarchy**, which contains a list of scene objects in the currently loaded scene. This panel can be used for **adding objects to scenes**, **modifying parent-child relationship** between objects (by drag-and-dropping) and **selecting objects**.
3. **Resources**, which contains all the currently loaded assets and created scenes. User can also **switch to different scene** through the resources panel.
4. **Inspector**, which contains modifiable properties of currently selected element (asset, scene, scene object, …)
5. **Scene**, which is used for building scenes. User can move around the scene, select objects and move them with gizmo manipulators. **For camera controls, see Section 3.**
6. **Event Graph**, which is used for building interactions with scenes. Each scene has its own Event Graph. **For Event Graph controls, see Section 5.**



The individual panels can be resized and docked next to each other, which allows the user to fully customize his work environment. Please note that this setup is currently **not persistent**, so exiting the application will reset the layout to its default state.

3. SCENE

You can move the **Editor camera** around the scene by **holding down** the **right mouse button**. While the button is down, you can look around the scene by panning the mouse. You can also navigate around the scene by keyboard **arrow keys**, or the **WASD** keys. You can elevate the camera by holding down **E** or de-elevate by holding down **Q**. All movement can be accelerated by holding down **Shift**.

While the right mouse button **is not down**, you can select scene objects by left clicking them. A selection outline will appear around most scene objects, indicating that they're selected. You can select multiple objects at once by holding down **Ctrl** or **Shift** while selecting an object. You can delete selected objects by pressing **Delete**.

The scene panel contains 5 buttons:

- **Play button** – This button will start/stop the Play mode of the currently loaded scene. When this button is pressed, the mouse cursor will be hidden. **To make the cursor reappear while in Play mode, press Alt**. You can also press **Escape** to exit Play mode

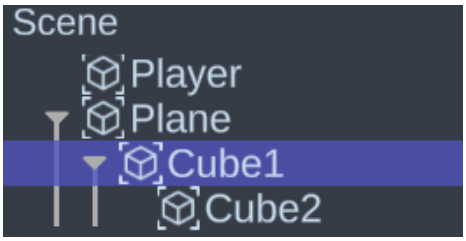
- **Translate button** (T) – This will make translation gizmo manipulators appear or hide. They appear as a set of three arrows. By clicking and dragging these arrows, you can **move the selected objects** around the scene.

- **Rotate button** (R) – This will make rotation gizmo manipulators appear or hide. They appear as a set of three circles. By clicking and dragging the mouse around the circle's circumference, the selected objects will **rotate around an axis**.

- **Scale button** (S) – This will make scale gizmo manipulators appear or hide. They appear as a set of long, thin cuboids. By clicking and dragging the manipulators, selected objects **scale in a particular axis**.

- **Space button** – This button toggles between a local object space and world space for the 3 mentioned gizmo manipulators. World space makes the gizmo manipulators point in world axes, which never change. Local object space depends on the object's current rotation.

4. HIERARCHY

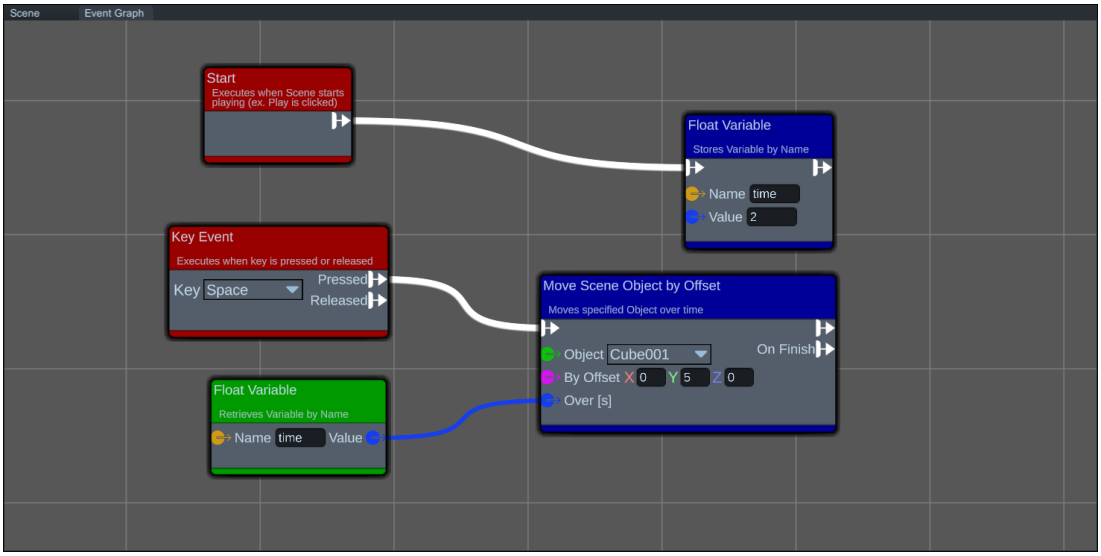


Hierarchy allows you to preview contents of the **currently** loaded scene. Scene objects form a hierarchical structure, meaning an object can have a parent object and set of child objects. You can change these relationships by **click-and-dragging** one element in between or on top of other objects. Hierarchy also allows you to **add** several predefined **objects** to your scene, which can be done using the **plus sign button** on top left of the hierarchy panel.

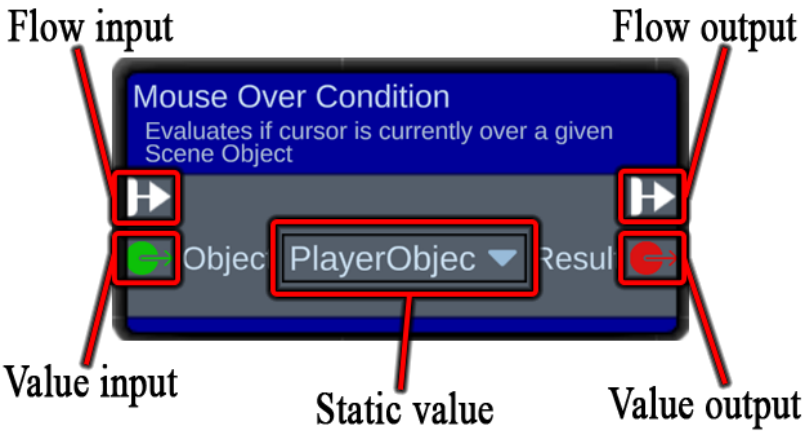
5. EVENT GRAPH

Event Graph is represented as a 2D plane with a grid pattern. You can **pan** around the event graph by **holding down** the **middle mouse key** and **dragging the mouse**.

Event Graph contains Nodes which serve as functions/data containers. You can **add Nodes** to the graph by **right-clicking** anywhere on this plane, which will make a context menu appear. You can select nodes by **left-clicking** on them. You can multi-select nodes by holding **Shift** while clicking on them. Nodes can be dragged around by mouse **click-and-drag**. You can duplicate selected nodes by pressing **Ctrl+D** or by **right-clicking** on a node and selecting the **Duplicate node** option.



Nodes may contain variable amount of **sockets**. Sockets come in 4 different types: flow input, flow output, value input and value output. By **click-and-dragging from the node's sockets**, you can **create connections** between nodes. You can **delete connections** with **right mouse button**.



Sockets determine 2 things: when the nodes are evaluated/executed and what values are used in evaluation. Node can receive data from other nodes via connections to **value input** sockets. If there's no connection, a default/static value is used. For certain data types, user can change these static values through the **static value** fields. Value socket connections can be only made between the same value data types. Data types are differentiated by color of the socket (and connection).
When a node is evaluated it stores the results in the **value output** sockets, which can then be used by other nodes through connection from the value output socket.

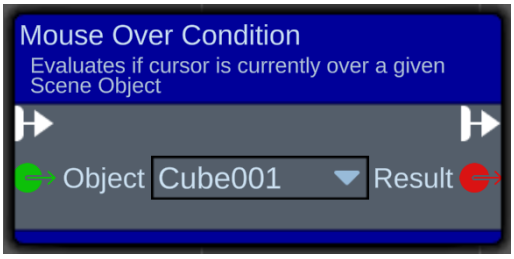
There are 3 main types of nodes:

1. **event nodes,**
2. **executable nodes,**
3. **non-executable nodes.**

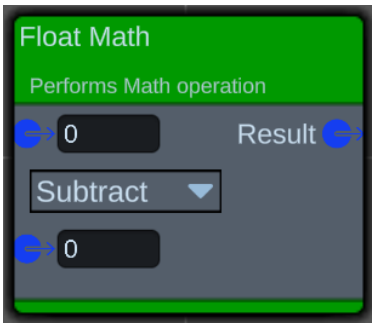
Event nodes **generate** flow control signals and deploy them through **flow output** nodes to other nodes in the Event Graph. Event nodes are usually indicated by **red color**.

Executable nodes are evaluated/executed, when they receive a control signal from connection to their **flow input** socket. Nodes can execute other nodes with **flow output** socket. This can happen after the

node is executed or mid-execution, depending on the specific type of node. Executable nodes are usually indicated by **blue color**.



Non-executable nodes are evaluated/executed every time a different node requests data value from its value input socket, that's connected to this node's value output socket. These types of nodes do not contain **any** flow input or flow output sockets. They are common for performing math calculations. Non-executable nodes are usually indicated by **green color**.



The node above would be evaluated every time a different node, connected to the "Result" value output would request the value. So in a sense, these nodes are evaluated in the opposite direction of the control flow.

6. NODES

This section contains a brief summary of the different types of available nodes.

Event nodes

These nodes **generate** flow control signals. The generation may depend on global scene events, user input, interaction of objects within a scene, etc.

Start node – generates a signal when the scene starts playing, meaning when user presses the Play button or when a scene is started through another scene's Event Graph.

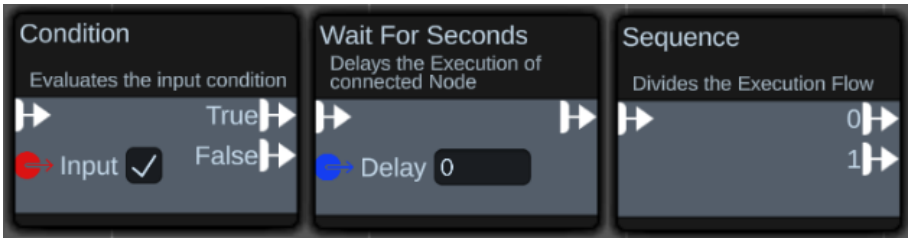
Update node – generates a signal on every frame, while the scene is running. Number of these updates per second is dependent on the framerate of the application.

Key event, Mouse event nodes – nodes generating signals when user presses or releases a particular mouse button or keyboard key.

Trigger event – generates a signal when a scene object **enters or leaves** the bounds of a particular **Trigger object** (Trigger Box or Trigger Sphere). The triggering object is also returned in a value output socket.

Signal Deployer and Receiver – nodes used for deploying and receiving signals identified by ID. Deployer node is not an event node per se, but works together with the Receiver node to form a functionality. When a Deployer node receives a flow control signal, all Receiver nodes sharing the same ID as the Deployer generate a flow control signal. If there are multiple Receiver nodes, the exact order of their relative execution is not defined.

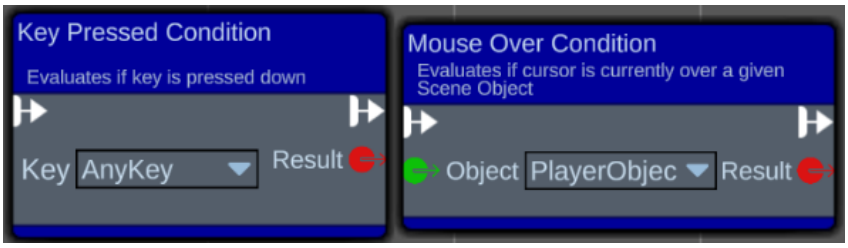
Flow Control nodes



These nodes are used to control the flow control signals.

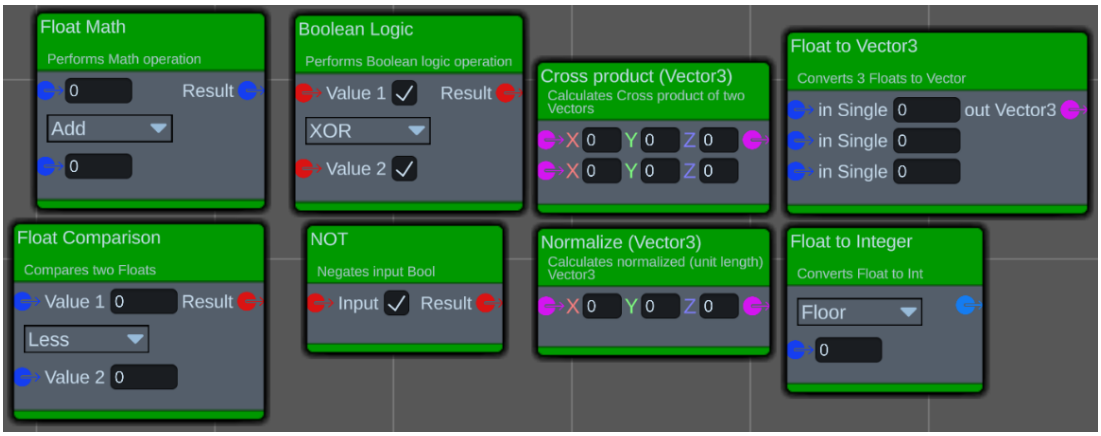
Condition node – when executed, it evaluates the boolean input value and depending on its value, it chooses which one of the two flow output sockets will deploy a control signal.
Wait For Seconds node – delays the execution of the next connected node by time specified in the delay value in seconds.
Sequence node – divides the flow control signal into two branches. The branches receive the signal in order specified by the number of the flow output.

Conditional nodes



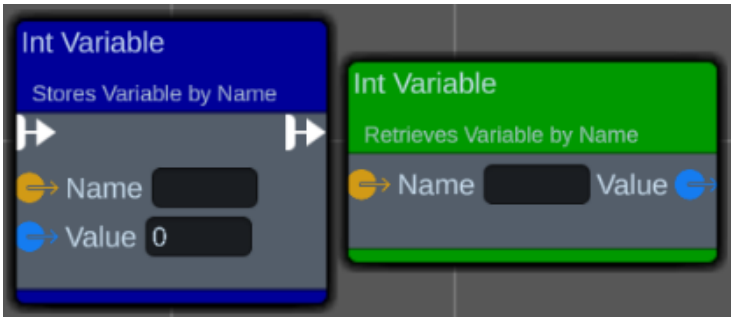
These nodes evaluate a particular condition when they receive a flow control signal and save the result in the output value socket. The result will be stored in the socket until the node is executed again, at which point the value will be re-evaluated and updated. **These nodes send flow output signals independently on the result of the condition.**

Math nodes



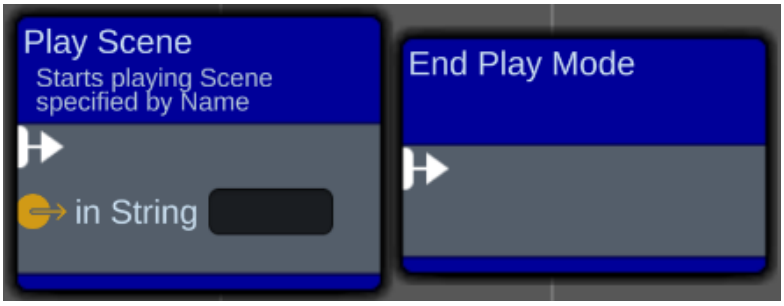
Nodes used for performing different math operations with common data types. This guide doesn't list all the different math nodes, as they all represent fairly common math operations. Math operations are supported for **float** values, **integer** values, **string** values, **boolean** values, **vector2** values and **vector3** values. They are also used for conversion between different types of data types.

Variable nodes



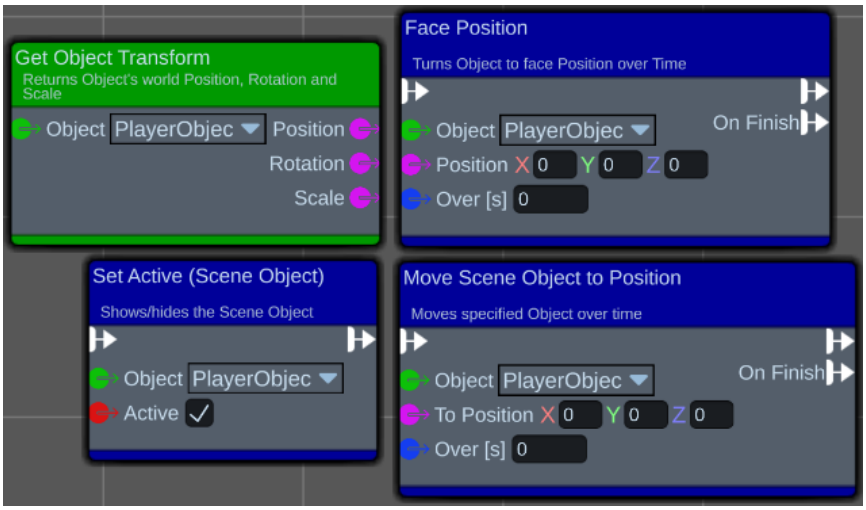
These nodes allow user to **store values** of certain data types and **retrieve** them when needed **during the play mode**, allowing them to save state. For example when user first enters a given section of the world, flag value is stored inside a bool variable node. Whenever the user enters that section, the flag value is retrieved and checked, allowing branching of the Event Graph flow depending on if it's the users first time entering or not. Variable node values are **not persistent through different scenes** and they are **reset whenever the play mode is stopped**.
For each data type that supports variables, there are two nodes. Setter – executable node used for storing the value to the variable, and Getter – non-executable node used for retrieving the stored value. Variables are identified by a **unique name**, which is determined by string value input.

Scene nodes



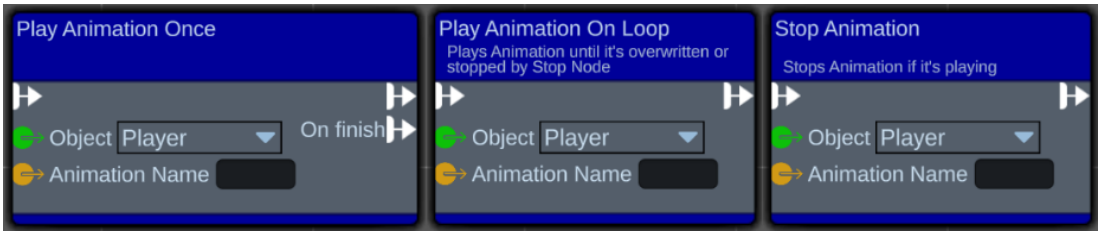
Nodes used for **switching to different scenes** in Play mode. This can also be used for restarting the current scene. The **End Play Mode node** exits the Play mode, stopping the story.

Scene Object nodes



Scene Object nodes, in general, retrieve information about specific Scene Objects, or perform operations with the objects, such as moving them to a different position or rotating them.
NOTE: At this stage of development, there are very few Scene Object nodes, as further research is required to figure out, which nodes should be implemented.

Animations



Imported 3D objects may contain animations. These animations are listed by their names in the **Inspector** panel. Animation nodes allow user to play and stop these animations at will. You just have to make sure that you type the animation name correctly. **Animation names, like all other identifying names, are case-sensitive**.

7. EXPORT FORMAT

Information in this section is only necessary if you need to understand exactly how the project save files are stored, for example, if you want to create a custom loader in different applications.

The saved information is contained inside a ".data" file. Every project file also has its associated asset folder, containing the user's imported data. So saving a project with filename **NewProject** will create a

file **NewProject.data** and folder **NewProject-Assets** in the selected directory. For loading, both the file and data folder **must be in the same directory**.

Special object types

The application contains several types of predefined special object types, that can be instantiated in scenes without importing any files. Some of these objects contain special fields.

Basic shapes

Empty object – Completely empty object with no geometry. Used for composition of objects using parent-child relationships.

Plane object – One sided square plane. Default size is 10 units wide and 10 units long.

Cube object – Contains simple 6-sided cube geometry. Default size is 1 unit in all 3 dimensions.

Sphere object – Contains simple sphere geometry. Default size is 1 unit in all 3 dimensions.

Cylinder object – Contains simple cylinder geometry. Default orientation is with the flat sides in the Y axis. Default diameter is 1 unit. Default height is 2 units.

Light object – Contains no geometry. Represents a point light in the scene. Light has Color, Range and Intensity properties.

Image object – Contains a both-sided plane geometry. Default orientation is with the plane's normal vector pointing in the negative Z axis. Users can display an image file on this plane. Plane has Width and Height properties. One world unit is equal to 1000 units of this size (meaning an Image object with Width and Height 1000 will be the same size as a unit cube).

Text object – Contains a both-sided plane geometry. Default orientation and size works the same way as the Image object. Additionally, the object contains Text property defining what is displayed, font size property and font color property.

Trigger Box and Trigger sphere objects – These objects contain only collision geometry which isn't visible in Play Mode. The default size is exactly the same as the Basic shape Cube and Sphere objects.

Player object – This object represents the Player when Play Mode is active. The camera attaches to this object and the object can be moved with user inputs. Player is a unique object for every scene, hence it can't be instantiated by the user. It also can't be set to Inactive, can't be deleted and can't be Parent of other Scene Objects. Player's collision is a capsule that's 1 unit in diameter and 2 units high by default. Additionally, Player has Collision property, defining if he should collide and be stopped by other Scene Object's geometry. If Collision is turned on, additional property is available, defining, if gravity force is applied to the Player.

Data file structure

Data file contains JSON serialized objects containing all the necessary data to load projects. They should also contain enough data so that users can create loaders in different applications by implementing their own renderers and Event Graph interpreters.

The data is separated into three main categories:

1. Scene Object Assets
2. Texture Assets
3. Scenes

Scene Object Assets is a JSON array. Each element has following fields:

`FileName` – Filename of the file in the assets folder, that contains geometry data of this asset.

`Name` – Name of the asset file displayed in the resources panel. Names do not have to be unique.

`AssetID` – Unique ID identifier of this asset. Other objects may reference this asset using this ID.

`ObjectData` – Contains serialized Scene Object data. This data serves as default values for when this asset is instantiated in scenes. For the format of Scene Object serialized data, see below.

Texture Object Assets is another JSON array. Each element contains has the following fields:

`FileName` – Filename of the file in the assets folder, that contains image data of this asset.

`Name` – Name of the asset file displayed in the resources panel. Names do not have to be unique.

`AssetID` – Unique ID identifier of this asset. Other objects or assets may reference this asset using this ID.

Scenes is the last JSON array. Each element represents one scene. Scene's structure is a bit more difficult and nested, because it contains data about every object in the scene, as well as the scene's serialized Event Graph.
Scene has following fields:

`ID` – Unique scene identifier. It may be referenced by other serialized data objects.

`Name` – Name of the scene, displayed in the resources panel. Doesn't have to be unique.

`EventGraph` – Serialized data of EventGraph (see below).

`SceneObjects` – JSON array of serialized Scene Objects (see below).

Scene Object contains the following fields:

`Type` – Type identifier. May be a regular SceneObject or one of the special types listed in the previous section, such as LightObject, TextObject, etc.)

`ID` – Unique Scene Object identifier. It may be referenced by other serialized data objects. If the object is not an instance (meaning it's only stored as an asset), the `ID` is null.

`Name` – Name of the scene object, displayed in the hierarchy panel. Doesn't have to be unique.

`SceneID` – ID of the scene in which this object belongs. Field is empty if this object isn't an instance in a scene.

`ParentID` – ID of the parent Scene Object. Field may be empty if this object is in the root of the hierarchy.

`IsFromAsset` – Boolean value defining, if this object is instantiated from Scene Object Asset.

`AssetID` – If `IsFromAsset` is true, this contains the `ID` of the Scene Object Asset from which it's instantiated, otherwise empty.

`IsInstance` – Boolean value defining, if this object is an instance in a specific scene.

`IsActive` – Boolean value defining, if this object is currently active. Non-active objects are not rendered and can't be interacted with.

`LocalPosition` – Position of object in relation to parent.

`LocalEulerRotation` – Rotation of object in euler angles in relation to parent.

`LocalScale` – Scale of object in relation to parent.

`Renderers` – JSON Array of Scene Object Renderers. Single Scene Object may be composed of multiple submeshes with multiple materials.

Scene Object Renderer represents one material and contains the following fields:

`Mesh` – Contains name of the submesh this renderer/material is attached to.

`MainColor` – Color of the material. Contains 4 float fields for the RGBA color components.

`TextureAssetID` – Field containing the ID of Texture Asset this material uses as source for texture. If material doesn't use textures, the field is empty.

`Metallic` – The metalness value of material based on PBR pipeline. Values range from 0 to 1.

`Smoothness` – The smoothness value of material based on PBR pipeline. It's the opposite of the more commonly used roughness property. Values range from 0 to 1.

`Offset` – This field defines the offset of the used texture, in other words, translation of the texture in the UV space. For that reason, the value is a 2-dimensional vector. Values of the vector should be in range from 0 to 1, as the offset just repeats itself beyond those values, since the wrapping mode of textures is set to Repeat.

`Tiling` – This field defines the tiling of the used texture, in other words, scaling of the texture in the UV space. For that reason, the value is a 2-dimensional vector. Values of the vector can be any floating point number.

Furthermore, special **scene objects** such as light or player **contain additional fields**. These fields are self-explanatory and follow the specification defined in the "Special object types" subchapter of this chapter.

Event Graph contains the following fields:

`SceneID` – ID of the scene this Event Graph belongs to.

`Nodes` – JSON array of Event Graph Nodes (see below).

`Connections` – JSON array of Event Graph Connections (see below).

Node represents a single instance of node in the Event Graph. It has the following fields:

`Type` – Contains string defining the node type. The most important node types are listed in Chapter 6 of this document.

`ID` – Unique node identifier within a single Event Graph. It may be referenced by Connections.

`Position` – Position on the Event Graph plane, value is a 2 dimensional vector.

`ValueInPorts` – Array of ports, each representing one **Value input socket**. Each element contains following fields:

- `ID` – Unique numeric index within the node.
- `ValueType` – Data type of the Value input.
- `Label` – Label of the socket. This is not needed for deserialization.
- `Value` – **Static value** stored in the socket. May be empty.

`ValueOutPorts` – Array of ports, each representing one **Value output socket**. Each element contains ID, ValueType and Label fields, which have the same meaning as fields of elements of ValueInPorts array.

`FlowInPorts` – Array of ports, each representing one **Flow input socket**. Each element contains ID and Label fields with the same meaning as these fields in elements of ValueInPorts array.

`FlowOutPorts` – Array of ports, each representing one **Flow output socket**. Each element contains ID and Label fields with the same meaning as these fields in elements of ValueInPorts array.

`EnumDropdowns` – Array of Dropdowns. Each element represents a Dropdown inside the node and contents of the Dropdown are defined by a public Enum. Each element in this array contains following fields:

`ID` – Unique numeric index within the node.

`Label` – Label of the dropdown. This is not needed for deserialization.

`EnumType` – String defining the Enum class name used to fill the Dropdown.

`Selection` – The currently selected Enum value.

Connection represents a single connection between sockets of two different nodes. It contains the following fields:

`ConnectionType` – Defines the type of connection. Can be either “Flow” or “Value”.

`NodeFromID` – ID of the node, from which the connection starts **(connection starts in output sockets and ends in input sockets)**.

`FromPortID` – Numeric ID of the output socket. Needed to fully determine from which socket the connection starts inside the specific node.

`NodeToID` – ID of the node, where the connection ends.

`ToPortID` – Numeric ID of the input socket. Needed to fully determine in which socket the connection ends inside the specific node.