VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D HRA S TECHNOLOGIÍ LEAP MOTION

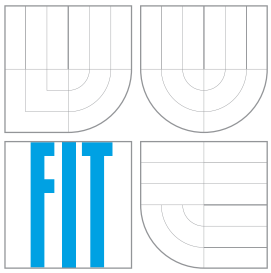
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATĚJ MAINUŠ

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D HRA S TECHNOLOGIÍ LEAP MOTION

3D GAME WITH LEAP MOTION CONTROLLER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATĚJ MAINUŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL, Ph.D.

BRNO 2014

Abstrakt

Cílem této bakalářské práce bylo navrhnout a implementovat 3D hru labyrint ovládanou pomocí gest rukou. Pro rozpoznávání pohybu a gest hra využívá technologii Leap Motion, aplikace samotná je vytvořena v herním enginu Unity. Výsledkem práce je multiplatformní 3D hra s vlastní knihovnou, která integruje Leap Motion SDK do Unity a eliminuje chyby v detekci rukou.

Abstract

The goal of this bachelor's thesis is to design and create a 3D labyrinth game controlled by hand gestures. This is achieved by using the Leap Motion sensor. Results of this thesis are the cross-platform 3D game based on Unity engine and a custom library that integrates Leap Motion SDK to the Unity engine. The developed library is able to suppresses detection errors introduced by the common LeapMotion SDK.

Klíčová slova

Leap Motion, ovládání pohybem rukou, ovládání gesty, Unity, C#, hra

Keywords

Leap Motion, hands motion control, gestures control, Unity, C#, game

Citace

Matěj Mainuš: 3D hra s technologií Leap Motion, bakalářská práce, Brno, FIT VUT v Brně, 2014

3D hra s technologií Leap Motion

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla, Ph.D.

.....
Matěj Mainuš
19. května 2014

Poděkování

Rád bych poděkoval Ing. Michalu Španělovi, Ph.D. za věcné připomínky a vedení práce.

Obsah

Kapitola 1

Úvod

Tématem mé bakalářské práce je návrh a realizace 3D hry s technologií Leap Motion. Toto zadání jsem si vybral, protože Leap Motion je nové zařízení o kterém dosud není napsáno mnoho prací a jedná se o velmi zajímavý senzor využitelný právě ve hrách. Cílem mé práce je vytvořit 3D hru labyrint ovládanou pohybem rukou s využitím zařízení Leap Motion a za pomoci herního enginu Unity. V aplikaci je kladen důraz na jednoduchost a přesnost ovládání, na grafické zpracování a přenositelnost mezi platformami. Součástí práce je vlastní knihovna, která integruje Leap Motion SDK do Unity, přidává podporu v ovládání GUI, eliminuje falešné objekty a odchytává stav kdy zařízení nekorektně rozpoznává orientaci objektů.

V úvodní kapitole jsou popsány dostupné možnosti detekce rukou a jejich gest. V následující kapitole je podrobněji popsáno zařízení Leap Motion, dále pak práce pokračuje kapitolou o herním enginu Unity. Kapitola o návrhu aplikace popisuje také herní principy, ovládání, fyziku ve hře a návrh vlastní knihovny rozšiřující Leap Motion API. Následuje kapitola o implementačních detailech skriptů, postupu tvorby scén, modelů a textur. Na konci práce jsem popsal testování aplikace a nastínil možnosti další vývoje.

Kapitola 2

Možnosti detekce gest rukou

Smyslem této kapitoly je popsat, jakými dostupnými způsoby lze detekovat pohyb rukou a jejich gesta. Tato kapitola popisuje speciální zařízení pro detekci gest, jako jsou Kinect 2.3, Wiimote 2.2 nebo Leap Motion 2.4, jejich přednosti, výhody a nevýhody.

Cílem této bakalářské práce, bylo vytvořit 3D hru, která je ovládána pohybem rukou a gesty. Ty lze snímat pomocí kamery, nebo nějakým speciálním zařízením jako je například Kinect nebo Leap Motion. Tyto zařízení kombinují více kamer snímajících prostor v různém spektru, proto je snímání objektů těmito zařízeními přesnější, než obyčejnou kamerou. K těmto zařízením je k dispozici sada nástrojů, které urychlují vývoj aplikace, protože autor nemusí řešit detekci, klasifikaci, a sledování objektů. Existují i zařízení, které pracují na jiném principu než je rozpoznávání obrazu, jako je tomu například u ovladače Wiimote nebo u teprve vyvíjeného ovladače Myo [1].

2.1 Kamera

Snímání obrazu kamerou, detekce rukou v obraze, a následné rozpoznávání a klasifikace gest je základní způsob, jaký použít k řešení problému ovládání aplikace gesty. Výhodou tohoto řešení je, že k němu není potřeba žádný speciální a drahý hardware. U tohoto řešení je ale nezbytné použít algoritmy pro detekci rukou v obraze a klasifikaci gest, které popisují například práce[7][13][1]. Detekce a klasifikace pak probíhá z důvodu složitých algoritmů na velmi malém počtu snímků za sekundu, což může mít vliv na latenci ovládání ve hrách. Při snímání kamerou s nízkým rozlišením, může být detekce nepřesná a při použití pouze jedné kamery nelze přesně určit pozici objektu v prostoru. Z tohoto řešení však vycházejí speciální zařízení jako je například Kinect.

2.2 Wiimote

Wiimote je skupina ovladačů ke konzolím Wii od společnosti Nintendo. Tyto zařízení má uživatel v ruce, vypadají dálkový ovladač a obsahují tří směrový akcelerometr a kameru. Akcelerometrem lze detekovat velmi výrazné pohyby rukou, kamera pak slouží pro určení pozice vůči konzoli[14]. Ovladač se však nikdy oficiálně nepoužíval jinak, než s konzolemi Nintendo.

[1] https://www.thalmic.com/en/myo/

Obrázek 2.1: Hra ovládána senzorem Wii[2].

2.3 Kinect

Kinect je senzor pro snímání pohybu lidí od společnosti Microsoft, původně navržený jako ovladač ke konzoli Xbox, později však bylo vydáno SDK i pro platformu Windows. Tato technologie je založena na kameře, hloubkovém infračerveném senzoru a speciálním čipu pro urychlení výpočtů. Výstupem ze zařízení jsou informace o kloubech kostry až dvou lidi[3]. Pro jeho původní zaměření je Kinect využíván často ve ve hrách, kde jako ovladač, slouží hráčovo tělo. Povětšinou se pro ovládání používají velmi výrazné pohyby a gesta, jako například mávnutí rukou, výskok, kopnutí, ... Oproti zařízení Leap Motion nedokáže sledovat pohyby dlaní a prstů, z důvodu nízkého rozlišení hloubkové kamery[12].

Obrázek 2.2: Ovládání hry senzorem Kinect[4].

2.4 Leap Motion

Leap Motion je speciální snímač uvedený na trh v roce 2013, který je navržený pro snímání rukou, prstů nebo také různých předmětů, jako jsou například tužky nebo pera. Snímač se skládá ze dvou infračervených kamer a tří diod. Jeho výhodou je přesné prostorové snímání[17] dvou rukou s rychlostí až dvě stě snímků za sekundu[6]. Pro vývojáře je k dispozici multiplatformní SDK které podrobněji popisuje kapitola 3. Vývojáři mohou své aplikace s navrženým nebo upraveným ovládáním pro Leap Motion umisťovat do aplikačního obchodu Airspace[2].

Obrázek 2.3: Součásti zařízení Leap Motion.

2.4.1 Ovládání aplikací pomocí snímače Leap Motion

Na vývojářských webových stránkách[3] jsou zveřejněny směrnice pro ovládání menu, tudíž všechny aplikace by se měly těmito pokyny řídit. Ovšem existují i výjimky, jako jsou aplikace určené k ovládání desktopového uživatelského rozhraní. Na ovládání hry nejsou kladeny žádné nároky, ale existují dva hlavní způsoby ovládání. Jsou ovšem k vidění i aplikace, které lze ovládat i pomocí nástrojů, jako jsou pera nebo tužky.

- První skupinou jsou aplikace ovládané pouze gesty. U těch nezávisí na poloze ruky, ale na provedeném gestu. Pro ovládání se používají velmi výrazná, dobře detekovatelná gesta, jako například švihnutí rukou, nebo virtuální dotek obrazovky
- Do druhé skupiny spadají aplikace ovládané polohou nebo náklonem ruky. U nich se používají spíše mírnější gesta jako například uchopení předmětu, kliknutí prstem, sevření ruky v pěst... Existují i aplikace využívající kombinace ovládání senzorem Leap Motion a klávesnice.

[2] https://airspace.leapmotion.com/
[3] https://developer.leapmotion.com/documentation/csharp/index.html

Kapitola 3

Leap Motion SDK

Tato kapitola popisuje základní aspekty práce s Leap Motion SDK, Jejím cílem je popsat hlavní a důležité části API, naznačit jakým způsobem s ním pracovat a ukázat jaké informace lze ze zařízení získat a k čemu je využít. Není cílem této kapitoly popsat všechny části, spíše jeho vlastnosti, výhody a nevýhody které se odráží v návrhu ovládání aplikace. Kapitola čerpá z oficiální dokumentace[5].

Sekce 3.1 popisuje strukturu API. V Části 3.2 jsem naznačil jakým způsobem se zpracovávají snímky. Sekce 3.3 a 3.4 se zaměřují na práci s detekovanými objekty, 3.5 na práci se základními gesty a v poslední části 3.6 jsou popsány problémy a nedostatky SDK.

Leap Motion SDK je oficiální programovací rozhraní pro Leap Motion. Lze ho stáhnout jako registrovaný vývojář z vývojářských stránek[1]. SDK je dostupné pro operační systémy Windows, Linux a OSX, platformy x86 a x86_64. Jeho součástí je dokumentace, sada příkladů a knihoven pro jazyky C/C++, C#, Java a Python. Pro další jazyky lze použít komunitní knihovny které lze najít také na vývojářských stránkách. Zařízení ke svému chodu vyžaduje speciální software, který je dostupný na stránkách projektu. Jeho součástí jsou ovladače pro senzor, vizuální debugger, aplikace obchodu Airspace a software pro nastavení a kalibraci senzoru.

3.1 Struktura API

API Leap Motionu lze rozdělit do čtyř celků.

1. Práce se zařízením
2. Modely detekovaných objektů
3. Gesta
4. Ostatní (vektory, matice, posluchače, utility)

Jednotlivé části API budou popsány v následujících kapitolách.

3.2 Detekce a klasifikace objektů

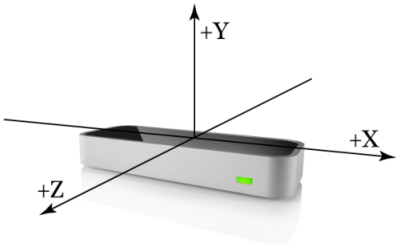
Detekce a klasifikace objektů se provádí neustále a na pozadí, takže do ní programátor přes svou aplikaci nemůže zasahovat. K zaznamenaným snímkům lze přistoupit přes třídu

[1] https://developer.leapmotion.com/

Controller, která obsahuje seznam šedesátí posledních snímků. Každý snímek je nezávislý na ostatních a obsahuje seznamy detekovaných rukou, prstů a předmětů. Do této třídy je možné zaregistrovat posluchače, který je informován o dokončení zpracování dalšího snímku a o odpojení nebo připojení snímače k počítači.

3.3 Reprezentace rukou

Detekovaná ruka je reprezentovaná třídou *Hand*. Ta obsahuje unikátní číselný identifikátor, kterým je možné provázat ruku mezi snímky, seznam detekovaných prstu a nástrojů, které jsou s rukou spojeny. Také je možné získat vektor vzdáleností dlaně od kontroléru, její jednotkový normálový i směrový vektor, vektor zrychlení a také dobu po jakou je ruka detekovaná. Vektory jsou reprezentovány v milimetrech a pravorukém kartézkém souřadném systému, který je zobrazen na obrázku 3.1

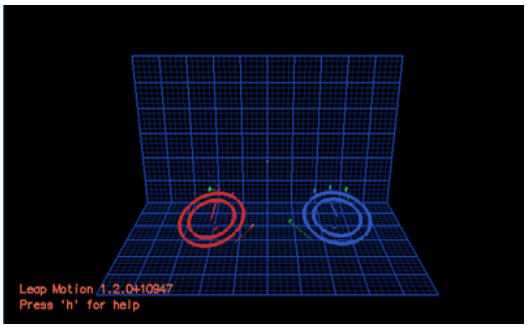


Obrázek 3.1: Souřadnicový systém v Leap Motion API[5].

3.4 Reprezentace prstů a nástrojů

Třídy *Finger*, *Tool* reprezentující prst a nástroj, jsou si velice podobné. Obě jsou potomky třídy *Pointable* a ta obsahuje prakticky všechny zásadní vlastnosti jako vektor pozic ve scéně, jednotkový vektor orientace, vzdálenost od ruky ke které patří, zrychlení oproti předchozímu snímku a obdobně jako v případě ruky také číselný identifikátor, který je unikátní pro každý prst nebo nástroj v rámci scény. Pokud uživatel sevře ruku v pěst, dojde k zmizení všech prstů. Když ji poté rozevře, Leap Motion se pokusí přidělit prstům stejné identifikační čísla, jako měly předtím. Grafickou reprezentaci dat ze snímače znázorňuje obrázek 3.2.

(a) Vizualizér dodávaný s Leap Motion SDK.

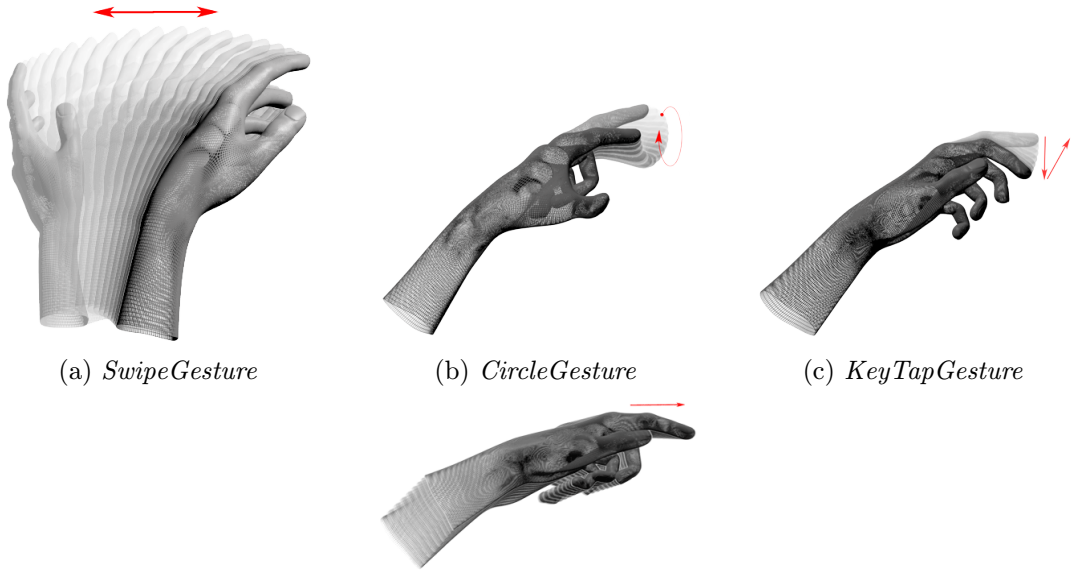


(b) Vizuální debugger pro Leap Motion.

Obrázek 3.2: Vizuální reprezentace dat ze senzoru Leap Motion

3.5 Gesta

Leap Motion API obsahuje základní sadu tříd pro rozpoznávání gest, jako je švihnutí rukou nad senzorem, dotyk virtuální plochy, nebo gesto kruhového pohybu prstu. Jednotlivá gesta a jejich třídy zobrazuje skupina obrázků 3.3. Všechny tyto třídy jsou potomky třídy *Gesture*. Ta obsahuje vlastnosti jako dobu trvání gesta, seznam rukou, prstů případně nástrojů které dané gesto provedly, typ gesta a jeho stav. Rozpoznávání těchto základních gest se musí aktivovat v třídě *Controller*.



(a) *SwipeGesture* (b) *CircleGesture* (c) *KeyTapGesture*

(d) *ScreenTapGesture*

Obrázek 3.3: Obrázky a třídy reprezentující gesta v Leap Motion API[5].

3.6 Problémy a nedostatky

Tato část se bude zabývat nedostatky a problémy Leap Motion SDK. Tyto problémy a nedostatky většinou pramení z problémů při detekci a klasifikaci. Nepopisuji zde všechny problémy, ale pouze ty, které jsou významné a důležité pro tuto práci. Praktické ukázky níže popsaných problémů lze vidět na videích, které jsou součásti přílohy A.

Záměna identifikačních čísel

První velmi důležitý problém je zmizení a znovunalezení objektu. Všechny detekované objekty, ať se jedná o ruce, prsty nebo nástroje, obsahují unikátní identifikační číslo. Jakmile dojde ke zmizení nějakého objektu, byť na velmi krátkou dobu, tak se ve většině případů stává, že mu po znovunalezení kontrolér přidělí jiné identifikační číslo než měl objekt předtím. U rukou se tento problém projevuje tak, že je ruce přiděleno úplně nové identifikační číslo, u prstů jedné ruky je situace poněkud horší. Jak bylo popsáno v sekci 3.4, v případně znovunalezení prsů se jim Leap Motion snaží přidělit stejné id jako měly předtím. Když zmizí více než jeden prst, tak ve velkém množství případů dojde k prohození jejich identifikačních čísel.

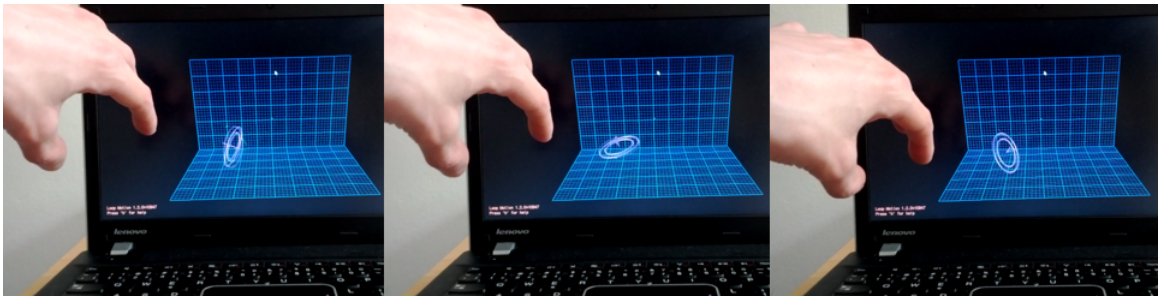
Detekce šumu

Druhým neméně častým, ale závažným, problémem je detekce šumu. Občas zařízení detekuje i objekty, které se nad ním nenacházejí. Toto chování nezávisí na tom, zda je nějaký reálný objekt již detekován. U těchto objektů také nejsou detekovány žádné prsty nebo nástroje a tyto falešné objekty také velmi rychle a prakticky neustále mění svou pozici, rotaci i směr, čehož lze využít při jejich eliminaci.

Obrázek 3.4: Vizuální debugger ukazující detekovaný šum.

Určení orientace

Pokud hráč nakloní ruku do pozice, ze které Leap Motion nedokáže po více snímků přesně určit orientaci, pak zařízení po tuto dobu poskytuje velice rozdílné hodnoty normálového a směrového vektoru. Toto chování lze vidět na sadě obrázku 3.5. Popsaná chyba může mít velmi negativní vliv na aplikace, které vyžadují velmi precizní ovládání odvozené z těchto údajů a je obzvlášť nepříjemná, protože se velmi špatně reprodukuje a ladí.



Obrázek 3.5: Ukázka špatného určení orientace ruky v průběhu času.

Další problémy, které ovšem nejsou důležité pro tuto práci popisuje L. E. Potter[10]. Jeho práce se věnuje hlavně problémům s rozpoznáváním prstů, jejich detekci a identifikaci.

Kapitola 4

Tvorba 3D her v Unity

V této kapitole jsou popsány vlastnosti a možnosti software Unity. Cílem této kapitoly je seznámit čtenáře s prvky enginu, které jsem využil v této bakalářské práci, popsat jeho výhody a vlastnosti. Při tvorbě hry jsem vycházel z oficiální dokumentace k Unity[16].

Části této kapitoly popisují práci s scénou 4.1, zdroji 4.2, možnosti skriptování 4.3, fyzikální engine který je součástí Unity 4.4, a tvorbu GUI 4.5.

Unity je souhrnný název pro editor a multiplatformní herní engine. Tento software je dostupný pod dvěma licencemi. Verze Unity Free neobsahuje všechny funkce a lze ji použít pouze k nekomerčním účelům, nebo u firem s obratem do 100 000 USD. Tuto verzi lze stáhnout zdarma z webových stránek[1]. Druhou verzí je Unity Pro, ta je dále rozdělena podle cílových platforem. Tyto verze jsou sice placené, ale obsahují všechny funkce, jejich porovnání lze najít na webových stránkách[2].

Obrázek 4.1: Unity editor.

[1] https://unity3d.com/unity/download
[2] https://unity3d.com/unity/licenses

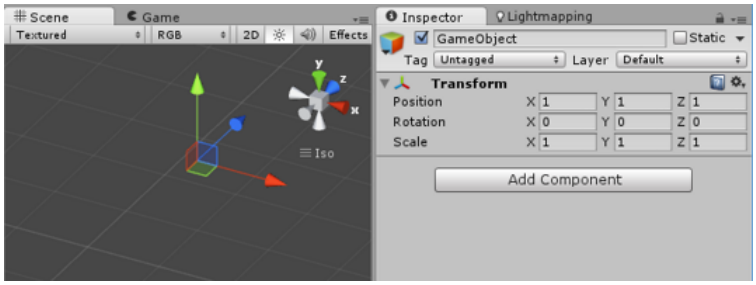
4.1 Scéna

Každý projekt v Unity je tvořen ze scén. Scéna je prostor, do kterého se vkládají objekty, světla, zdroje zvuku, kamery, spouštěcí události atd . . . . V průběhu hraní hry se pak přechází mezi jednotlivými scénami.

Rodičem každého prvku scény je *GameObject*. Ten obsahuje základní informace o objektu, jako jsou pozice, rotace a zvětšení. Každému objektu lze přiřadit další moduly, jak je například renderer, kolizní modely, nebo lze z něj udělat světlo, kameru, zdroj zvuku, ale také lze na něj aplikovat skupinu skriptů.

Unity používá pro umístění modelů do scény levoruký kartézský systém souřadnic, zobrazeným na obrázku 4.2. Jejich pozici a zvětšení ukládá v tříprvkovém vektoru, rotaci ve stuktuře quaternion,

Objektu lze přiřadit značku, podle které s ním lze pracovat ve skriptech, případně ho zařadit do vrstev, které slouží například pro omezení působnosti světel a výpočtu stínů, nebo pro omezení výpočtu kolizí, mezi objekty v různých vrstvách.



Obrázek 4.2: Systém souřadnic v enginu Unity.

4.2 Zdroje

V Unity se nazývají všechny externí objekty jako zdroje, ať se jedná o model, texturu, materiál, skript, animaci, skin pro GUI nebo prefab. Zdroje jsou většinou staticky vloženy do scény, existují ale také typy zdrojů, které lze dynamicky za běhu aplikace načítat a odstraňovat. Po vytvoření aplikace jsou zdroje zakomponovány do speciálního balíku, ze kterého jsou poté načítány. Verze Pro disponuje vlastností, dovolující načítat i zdroje mimo tento balík.

Do Unity lze vložit model a materiály z prakticky všech modelovacích nástrojů. Interně pro převod modelů z většiny těchto nástrojů se používá formát FBX. Přestože lze importovat i materiály, je lepší je vytvářet přímo v Unity, protože materiály jsou zde velice úzce svázány s shadery. Standardní shadery, které jsou k dispozici v Unity jsou rozděleny do těchto pěti rodin: *Normal*, *Transparent*, *Self-Illuminated*, *Reflective* a *Mobile*. V každé skupině se pak nachází některý z shaderů těchto typů: *Vertex-Lit*, *Diffuse*, *Specular*, *Bumped*, *Parallax* a jejich kombinace. Výběr shaderů, který se použije pro rendering materiálu, má velmi velký vliv na rychlost renderování. Proto je velmi důležité vybírat shadery s ohledem na vlastnosti materiálu, vyhýbat se co nejvíce shaderům z rodiny *Transparent*, protože u nich musí probíhat alfa test. V kritických situacích je vhodné používat rychlejší, ale méně sofistikované shadery ze skupiny *Mobile*. U objektů, které jsou vzdálené od kamery, nebo nejsou tak často vidět, není nutné používat drahé shadery s normálovou mapou, která na nich nepůjde poznat. Unity používá pro psané shaderů jazyk Cg od společnosti Nvidia.

Unity podporuje 2D a 3D textury všech často používaných formátů. Textury je vhodné vkládat s rozměry, které jsou mocniny dvou, jinak je Unity na tyto rozměry transformuje, což je v některých případech nežádoucí. U textur které se používají jako normálové mapy, je vhodné je takto importovat, protože pak dochází k optimalizacím těchto textur. Normálovou texturu lze v Unity vygenerovat také z bump mapy.

Jedním speciálním zdrojem je takzvaný Prefab. Tyto zdroje se vytváří přímo v Unity a slouží jako před vytvořené komponenty, skládající se z více GameObjektů. Tyto prefaby lze pak používat ve více scénách. Jestliže pak dojde ke změně parametrů nějakého objektu v prefabu, tak se tyto parametry promítnou do všech objektů, které z tohoto prefabu vycházejí.

4.3 Skripty

Skriptovat lze v jednom z jazyků C#, Javascript nebo Boo. Každý objekt pak může obsahovat více skriptů, které řídí jeho chování. Tyto skripty obsahují třídu která je potomek třídy *MonoBehaviour*. Přetížením funkcí této rodičovské třídy lze s objektem manipulovat pomocí velmi rozsáhlého API.

Všechny použité skriptovací jazyky obsahují automatický správce paměti. Ten ale může být zdrojem velkých výkonnostních problému, pokud se ve skriptech provádí velmi často obrovské množství alokací. Například v jazyce C# je kritické například časté spojování řetězců, používání funkcí z LINQ, techniky zvané boxing, ale také použití foreach smyček. Ty v implementaci běhového prostředí Mono, která je součástí Unity, využívají pro průchod instanci rozhraní IEnumerator alokovanou na hromadě, která je spravována právě automatickým správcem paměti[11]. Tyto způsoby mohou mít velký vliv na výkon, pokud se provádí často například před nebo po vykreslení každého snímku.

4.4 Fyzika

Pro simulaci fyziky používá Unity fyzikální engine PhysX vyvíjený společností Nvidia. Jedná se ale o starší verzi 2.8 z roku 2007[9]. Tento engine je do Unity velmi dobře integrován a na první pohled není vůbec vidět. Simulace fyziky je nedílnou součásti každé scény a nelze ji vypnout. Jednotlivým objektům ve scéně se přidávají komponenty, které určují fyzikální vlastnosti objektu. Lze do něj přidat skupinu kolizních modelů, vytvořit z něj tuhé těleso, aplikovat na něj konstantní sílu nebo ho spoutat s jiným modelem. Simulace fyziky se provádí pro každý objekt v předdefinovaných časových intervalech, jestliže se však objekt nehýbe je tento interval prodloužen. Koliznímu modelu lze také přiřadit fyzikální materiál, který určuje jeho statické a dynamické tření a způsob jejich výpočtu při kolizi s jiným objektem.

4.5 GUI

Součástí Unity je taky sada API pro práci s GUI. Jeho princip je založen na periodickém vykreslování elementů v předem stanovenou dobu. Grafické prvky se vytvářejí volání statických funkcí které jako výsledek vracejí stav prvku. U tlačítek je to například zda je stisknuto, u textového vstupu jeho obsah, u posuvníků jejich hodnota atd.... Při kreslení se GUI chová jako stavový automat, čili když se změní nastavení hodnoty ve statické třídě *GUI*, všechny prvky vykreslené od tohoto místa dále, budou s touto hodnotou operovat.

API obsahuje velké množství grafických prvků jako jsou tlačítka, vstupní pole, zaškrtávací tlačítka, posuvníky atd.... Elementy se na obrazovku umisťují na přesně určenou pozici, nebo pomocí jednoduchých automatických rozložení. Vytvořené GUI lze přestylovat podle svých požadavků.

4.6 Optimalizace scén

Unity jako herní engine obsahuje velké množství optimalizačních funkci, jako jsou například Static Batching, Occlusion Culling, Deferred Lighting Rendering Path, LOD, Light Probes. Všechny tyto funkce jsou k dispozici až ve verzi Unity Pro a mají význam až u rozsáhlých projektů s velkými scénami, s mnoha objekty a světly, nebo u projektů cílených například na mobilní zařízení.

Kapitola 5

Návrh 3D hry labyrint ovládané pohybem rukou

Cílem této práce je vytvořit 3D hru labyrint ovládanou pohybem, náklonem a gesty rukou, s využitím zařízení Leap Motion. Tato kapitola popisuje návrh aplikace, její strukturu a technologie, které jsem využil k realizaci hry. Důležitým tématem návrhu je vlastní knihovna, jejíž cílem je odstranit nedostatky Leap Motion SDK popsané v kapitole 3.6 a integrovat ji do zvoleného herního enginu. Výsledkem práce bude multiplatformní aplikace, skládající se z menu a samotné hry.

Při vytváření hry jsem musel navrhnout řešení pro tyto části:

- Princip hry a její metriky
- Menu a scény
- Ovládání aplikace pohybem rukou
  - Ovládání menu
  - Ovládání náklonu herní desky
- Herní engine
- Fyzika

Detailní návrh řešení těchto aspektů je popsáno v následujících kapitolách.

5.1 Herní princip

Tato hra se inspiruje starou dřevěnou hrou labyrint, ve které je cílem dostat kuličku do správného důlku pomocí naklánění labyrintu. V mé modifikaci může uživatel hrát s více kuličkami najednou, přičemž hra náhodně určuje pořadí v jakém je nutné kuličky do důlku dostat. Labyrinty mohou také obsahovat více míst, ze kterých se kuličky umísťují do scény. Během hraní se uživateli počítá skóre a také měří čas, za jak dlouho je schopen umístit všechny kuličky do cílového důlku. Jestliže hráč umístí aktivní kuličku do správného důlku, přičte se mu ke skóre pět bodů. Pokud ovšem umístí neaktivní kuličku do libovolného důlku nebo umístí aktivní kuličku do nekoncového důlku, odečtou se mu ze skóre dva body. Umístěním poslední kuličky do správného důlku hra konči a přestává se měřit čas. Výsledky

hry se budou podle úrovní ukládat do souboru umístěném ve skrytém adresáři v domovské složce uživatele.

Hra bude obsahovat více modelů labyrintů a každý z nich lze hrát ve více konfiguracích, kde se mění celkový počet kuliček a počet aktivních kuliček. Zatím si bude možné vybrat pouze z přednastavených konfigurací. Labyrint se pak bude nacházet v jedné náhodně vybraných scén.

Tuto hru jsem si vybral proto, že je zde potřeba velmi precizní ovládání, které lze realizovat pomocí ovladače Leap Motion, ale je neuskutečnitelné na klávesnici nebo myši.

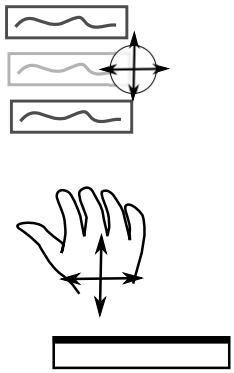
5.2 Ovládání hry pohybem rukou

Detekci rukou, jejich sledování budu provádět zařízením Leap Motion. Toto zařízení jsem si vybral proto, že je určeno právě pro snímání rukou, pomocí nichž se bude aplikace ovládat a poskytuje velmi přesné informace o jejich poloze a rotaci.

5.2.1 Ovládání menu

Mým cílem je vytvořit aplikaci tak, aby bylo možné ovládat jak menu tak naklánění hrací desky pohybem rukou a gesty. Pro neznalé uživatele jsem ale zachoval ovládání menu pomocí myši.

GUI se ovládá tak, že uživatel dá svou ruku nad zařízení a objeví se mu na obrazovce kurzor. Ten se hýbe podle polohy uživatelovi ruky. Interakci s tlačítky jsem nejprve realizoval gestem sevření ruky v pěst. Toto gesto se později ukázalo jako nevhodné, protože při sevření ruky uživatel rukou mírně pohne a to při malé velikosti ovládacích prvků způsobí kliknutí vedle. Také je toto gesto pro neznalé uživatele neintuitivní. Obdobný problém způsobuje také, pokud má uživatel klikat na tlačítka pohybem ruky vpřed. Proto jsem kliknutí na tlačítko realizoval časovačem. Jestliže uživatel najede kurzorem na prvek, dojde ke změně barvy kurzoru a spuštění časovače. Kurzor poté podle jeho hodnoty mění barvu, aby se indikoval čas do kliknutí. Kliknutí se provede jakmile časovač dosáhne jedné vteřiny. Jestliže kurzor opustí prvek před kliknutím dojde k vynulování časovače.



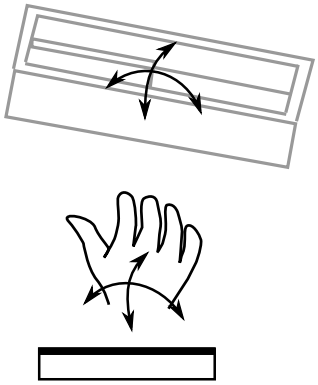
Obrázek 5.1: Náčrt principu ovládání menu.

5.2.2 Ovládání labyrintu

Ve samotné hře se ovládá pouze labyrint a to tak, že nakláněním dlaně se naklání i labyrint. Ten ovšem nelze naklánět pouze podle normálového vektoru ruky. ale je potřeba od něj

odečíst počáteční offset. To z důvodu, že hráč nikdy nedá ruku nad kontrolér tak, aby její normálový vektor mířil přesně dolů. Labyrint půjde také ovládat pomocí dvou rukou, které budou v horizontální poloze. To z důvodu, že Leap Motion velmi nepřesně rozpoznává orientace rukou pokud jsou ve vertikální poloze.

Maximální úhel natočení labyrintu bude omezen na 40°vůči vodorovné poloze, aby jej hráč nemohl otočit směrem dolů. V menu bude možné nastavit citlivost náklonu hrací desky.



Obrázek 5.2: Náčrt ovládání labyrintu jedou rukou.

5.3 Rozšíření Leap Motion API

Motivací pro vytvoření vlastní knihovny bylo, oddělit řešení problémů z Leap Motion od zbytku hry a vytvořit nad jeho API dokonalejší vrstvu, která se usnadní integraci do Unity. Jedná se o oddělenou sadu zdrojových souborů, které jsou součástí projektu, protože verze Unity Free podporuje externí knihovny velmi omezeně. Tyto zdrojové soubory je možné využít odděleně i v jiném projektu, případně z nich vytvořit dynamicky linkovanou knihovnu.

Hlavním úkolem knihovny bude integrovat Leap Motion API do Unity tak, aby zpracování snímků probíhalo v době, která je tomu enginem určena. Dále budou součástí knihovny třídy vytvářející obal nad detekovanými objekty pro jejich jednodušší použití v Unity, skript doplňující podporu v ovládání Unity GUI pomocí ovladače Leap Motion, třídu přidávající detekci gesta sevření ruky. Jak ukazují obrázky 3.1 a 4.2, Unity reprezentuje vektory v jiném souřadném systému než Leap Motion, proto knihovna obsahuje utilitu pro převod souřadnicového systému Leap Motion API do souřadnicového systému Unity.

5.3.1 Řešení nedostatků z Leap Motion SDK

Část 3.6 popisuje nedostatky Leap Motion SDK, proto jedním z důvod vzniku této knihovny byla motivace pro jejich částečné eliminování. Většina problémů totiž má velice významný vliv na hru, ve které se klade velký důraz na přesnost ovládání, která závisí hlavně na orientaci ruky a tu má občas senzor problém správně určit. V knihovně je proto definováno několik konstant, které vychází z mnou provedených experimentů, přesto však nejsou tyto metody vždy účinné a najdou se případy ve kterých nefungují.

Záměna identifikační čísel

Při zmizení ruky ze scény, nedojde k jejímu úplnému odstranění. Místo toho se spustí časovač, který po překročení hraníce 0,4s prohlásí ruku za neaktivní, a po 1,0s za neplatnou.

Tato technika zajistí, že pokud dojde ke krátkodobém zmizení, tak se ruka zbytku aplikace jeví stále jako aktivní.

Pokud se objeví nová ruka, projdou se všechny neaktivní ruce, a jestliže se najde ruka, která se pozicí, rotací a směrem, velice podobá té neaktivní, pak se původní ID nahradí identifikačním číslem nově nalezené ruky. Podobnost ruky je definována vzorcem 5.1, kde I je neaktivní ruka N je nově nalezená ruka. $\vec{p}$ je vektor pozic, $\vec{n}$ je jednotkový normálový vektor, $\vec{d}$ je jednotkový směrový vektor. Touto metodou je zajištěno, že při změně identifikačního čísla při výpadku se pro zbytek knihovny a aplikace nic nemění.

$$|\vec{p_I} - \vec{p_N}| < 20 \wedge \measuredangle \vec{n_I} \vec{n_H} < 15° \wedge \measuredangle \vec{p_I} \vec{p_N} < 15° \tag{5.1}$$

Detekce šumu

Zařízení občas prezentuje jako ruce i objekty které se nad ním fyzicky nenacházejí. Jak popisuje sekce 3.6 tyto objekty mají velice specifické chování. Proto pro odlišení musí hráč umístěnou ruku ponechat v klidu po dobu 0,5s, aby došlo k ověření, že se opravdu jedná o ruku. Klidový stav nastane, platí li výraz 5.2, kde $\vec{p}$ je vektor pozic, $\vec{n}$ je jednotkový normálový vektor, $\vec{d}$ je jednotkový směrový vektor, f je počet prstů a i je číslo snímku:

$$|\vec{p_i} - \vec{p_{i-1}}| < 2 \wedge \measuredangle \vec{n_i} \vec{n_{i-1}} < 3° \wedge \measuredangle \vec{p_i} \vec{p_{i-1}} < 3° \wedge f >= 2 \tag{5.2}$$

Eliminace špatného určení orientace

Nepřesná detekce, která nastává při nevhodné orientaci hráčovi ruky, má u této hry velmi významný vliv. Pokud dojde k velmi razantní změně v rotaci nebo směru, pak je tato změna dočasně ignorována, uložena, dále se vynuluje časovač určující, že ruka je stabilní a čeká se na vyhodnocení dalšího snímku. Tato změna se také neprezentuje ven pro zbytek aplikace. Jestliže se při vyhodnocení následujícího snímku zjistí další razantní změna, značí to problém popsaný v kapitole 3.6 a tento postup se opakuje. Pokud se v dalším snímku ruka chová klidně, připočítá se k časovači doba mezi posledním snímkem a aktuálním a poté co suma těchto časů dosáhne 0,2s, pak se zbytku aplikace prezentuje poslední orientace ruky. Tento algoritmus zajistí potlačení velkých a dlouhodobých změn, které pravděpodobně nezpůsobuje hráč.

5.3.2 Gesto sevření ruky v pěst

Knihovna bude také rozšiřovat základní gesta z Leap Motion SDK o gesto sevření ruky v pěst. Toto gesto budu detekovat podle počtu rozpoznaných prstů ruky. Protože Leap Motion nedokáže detekovat prsty správně, pokud je ruka v jiné, než vodorovné poloze, je detekce gesta sevření ruky v pěst tímto podmíněna. V případě hry se jedná o dobrou vlastnost, protože je toto gesto využito k pozastavení hry, které by mělo nastat, když je labyrint ve vodorovné poloze.

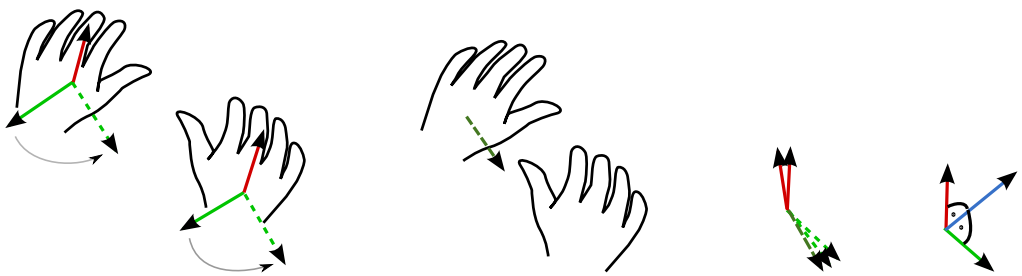
Jestliže počet rozpoznaných prstů klesne na jeden, je k časovači přičtena doba mezi posledním a aktuálním snímkem. Když v následujícím snímku dojde k rozpoznání více než jednoho prstu, je doba časovače snížena o jeden a půl násobku času mezi předchozím a aktuálním snímkem. Jakmile časovač dosáhne času 0,5s je gesto považováno za aktivní. Gesto je označeno z aktivního na neaktivní v případě že hodnota časovače klesne pod 0,2s.

5.3.3 Podpora Unity GUI

V Unity GUI je implementováno ovládání pouze pomocí myši, klávesnice a doteků. Jedná se o uzavřený systém, do kterého nelze systémově integrovat ovládání senzorem Leap Motion. Proto jsem se rozhodl vytvořit vlastní třídu implementující ovládání GUI zařízením Leap Motion, která bude mít velmi podobnou strukturu a sémantiku funkcí jako je tomu u prvků GUI z Unity. Při vytváření GUI se budou volat standardní funkce z Unity, k nim se přidá volání funkcí z mé knihovny, které přidají podporu ovládání o gesta rukou.

5.3.4 Ovládání pomocí dvou rukou

Rozhodl jsem se, že labyrint půjde ovládat pomocí dvou rukou. Cílem knihovny je poskytovat stejné údaje jako v případě ovládání jednou rukou. Proto jsem se vymyslel algoritmus, který podle rozdílu polohy v ose Y a dle normálových vektorů dlaní, váženým průměrem vypočítá vektor reprezentující rotaci podél osy Z. Průměrem směrových vektorů se vypočítá vektor určující rotaci kolem osy X. Vektorovým součinem těchto dvou vektorů reprezentující rotace v osách Z a X, dostaneme jeden normálový vektor, který reprezentuje rotaci obdobně jako kdyby uživatel ovládal labyrint jednou rukou. Schématický postup výpočtu je zobrazen na obrázku 5.3



Obrázek 5.3: Princip výpočtu normálového vektoru při ovládání dvěma rukama.

5.4 Menu, scény a labyrinty

Po zapnutí aplikace se uživateli zobrazí menu ve které budou následující položky:

- Nová hra - Po kliknutí na toto tlačítko se zobrazí uživateli mřížka s tlačítky reprezentující jednotlivé úrovně a tlačítko zpět. Po kliknutí na jedno z tlačítek s úrovněmi se spustí samotná hra s vybraným labyrintem.
- Výsledky - Vybráním sekce výsledky se zobrazí stejná mříž jako u nové hry, ale po zvolení jedné úrovně se zobrazí v seznamu pět nejlepších výsledků obsahující datum, dosažené skóre a čas za jaký byla hra dohrána. Výsledky budou seřazeny primárně podle skóre a sekundárně podle času.
- Nastavení - V této části si bude moci hráč nastavit citlivost naklánění labyrintu
- Konec - Ukončení aplikace

Spuštěním samotné hry bude zobrazena scéna se stolem, na kterém bude labyrint s kuličkami odpovídající zvolené konfiguraci. Jakmile hráč umístí jednu nebo obě ruce nad senzor

Leap Motion, dojde k spuštěním měřiče času a zvednutí labyrintu ze stolu. Ten se začne naklánět podle hráčových rukou. Když hráč odejme ruce z dosahu senzoru, čas ve hře se pozastaví a labyrint se položí opět na stůl. Hru lze také pozastavit sevřením ruky v pěst.

Po dokončení hry se uživateli zobrazí seznam nejlepších výsledků z dané úrovně, který bude seřazený primárně dle skóre a sekundárně dle času.

Jestliže hráč stiskne během hry klávesu escape, čas ve hře se pozastaví, dojde k položení labyrintu na stůl, a zobrazí se hráči menu s položkami:

- Pokračovat
- Hrát znovu
- Zpět do menu
- Konec hry

Při opětovném stisknutí klávesy escape, dojde ke schování menu.

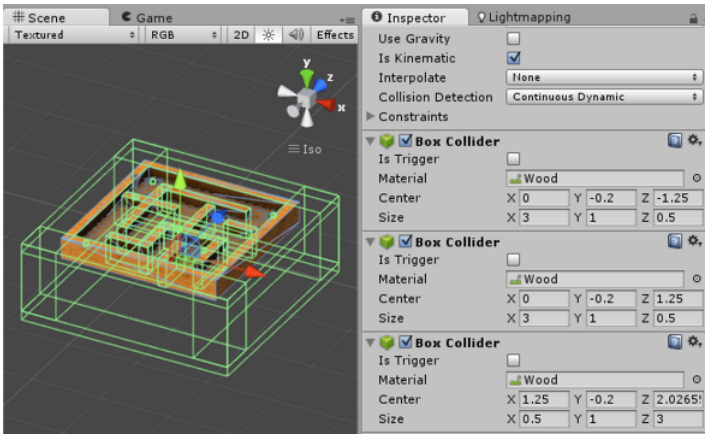
Hra bude obsahovat jeden ukázkový labyrint, na kterém si může hráč vyzkoušet ovládání a případně upravit citlivost. Ostatní labyrinty budou vytvořeny tak, že hra bude vyžadovat precizní ovládání. Cílem bude situovat důlky na místa, kde mohou hráči překážet a bude tak zvýšená možnost že se do nich občas trefí. Stěny uprostřed labyrintu budou umístěny tak aby dráha kuličky od počátku do konce nebyla příliš krátká, také nebude možné použít stěny v režimu s více kuličkami k odložení kuliček na bezpečné místo. Pro některé labyrinty bude existovat více předem určených startovacích pozic, ze kterých budou náhodně kuličky začínat.

5.5 Herní engine

Jako nástroj k vytvoření hry jsem si vybral engine Unity ve verzi Free a to proto, že se jedná o engine poskytovaný zdarma i pro malé komerční projekty, obsahuje široké API a skriptování se provádí ve vysokoúrovňových jazycích. Unity integruje v sobě velmi kvalitní fyzikální engine PhysX, má sadu předdefinovaných a dobře optimalizovaných shaderů, umí zobrazovat modely z velkého množství modelovacích nástrojů a obsahuje sadu nástrojů, které lze využít při optimalizacích hry.

5.6 Fyzika

Pro simulaci fyziky jsem použil fyzikální engine PhysX, který je dostupný přímo v Unity. I přesto nebyla implementace fyziky jednoduchá. V enginu se jedna jednotka rovná jednomu metru v realitě[16]. Tato vlastnost má velmi významný vliv na simulaci malých objektů, které jsou menších než 10 cm. Jelikož kulička v labyrintu je reálně menší než tato velikost, je nutné celý svět zvětšit tak, aby její velikost byla dostatečná. Proto jsem všechny modely vytvořil v desetinásobném zvětšení. Také jsem musel upravit gravitační zrychlení, aby se objekty ve scéně chovaly přirozeně. ... Jako kolizní model pro celý labyrint, jsem nemohl použít původně zamýšlený *MeshCollider*. Tento kolizní model aplikovaný na tuhé těleso totiž nekoliduje s ostatními tuhými tělesy pokud není bázový model konvexní, nebo z něj není spočítána konvexní obálka[16]. Proto se tento kolizní model používá pouze na některé části labyrintu. Ten se z většiny skládá z vice kolizních primitiv jako jsou například kvádry.



Obrázek 5.4: Fyzikální model labyrintu.

Fyzikální engine také slouží k detekci pádu kuličky do důlku. Každý důlek obsahuje virtuální kolizní objekt. Pokud kulička koliduje s tímto objektem, považuje se to za její pád do důlku.

Jako metodu výpočtu kolizí mezi kuličkou a labyrintem jsem využil spojitou detekci kolizí. To z důvodu malé velikosti kuličky vůči stěnám labyrintu. S diskrétní metodou docházelo k průletu kuličky stěnou. Tento problém je v známý pod anglickým názvem *Bullet through paper*[8] a se ještě výrazněji projevuje při pohybu dvou objektů proti sobě, což je případ naklánění labyrintu proti směru pohybu kuličky. Protože ani spojitá detekce kolizí neeliminovala tento problém úplně, zmenšil jsem velikost simulačního kroku na polovinu původní hodnoty a také jsem pro všechny venkovní stěny vytvořil větší kolizní modely aby nedocházelo k průchodu kuličky mimo labyrint. Také jsem se snažil navrhovat labyrinty tak, aby kulička nemohla dosáhnout potřebné rychlosti k průletu stěnou nebo podlahou.

Na rychlost pohybu kuličky mají vliv fyzikální materiály. Mým cílem bylo vytvořit hru co nejpřesněji odpovídají realitě, proto jsem podle materiálu objektu použil fyzikální materiál, který určuje statické a dynamické tření. Materiály které jsem využil jsou součástí Unity, pouze jsem u nich mírně upravil konstanty tření, aby se hra chovala reálněji.

Kapitola 6

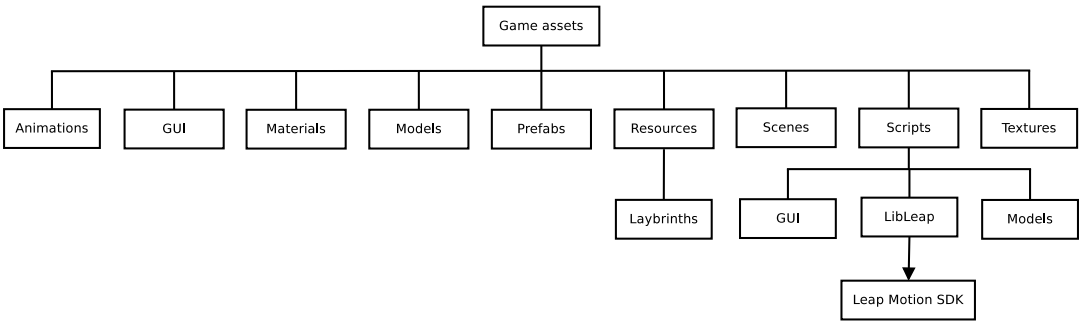
Implementace hry

Tato kapitola popisuje implementaci významných částí samotné hry labyrint v enginu Unity. Jejim cílem není detailně popsat všechny vlastnosti a funkce každého skriptu, spíše strukturu aplikace a účely jednotlivých skriptů.

Jako programovací jazyk pro skriptování v Unity enginu, jsem si vybral C#. Modely do hry jsem vytvořil v modelovacím nástroji Blender[1], základ pro textury jsem získal z webu CGTexures.com[2], které jsem upravoval v programu Krita[3].

6.1 Struktura

Výsledná aplikace se skládá z enginu Unity, knihoven pro Leap Motion a zdrojů pro aplikaci. Ty jsou rozděleny do složek podle typu na animace, zdroje pro GUI, textury, modely, materiály, scény a skripty. Ty jsou dále děleny na datové modely, skripty pro herní objekty, knihovnu rozšiřující Leap Motion SDK a skripty vytvářející menu. Oddělenou částí jsou zdroje pro labyrinty (modely, textury, materiály). To proto aby je bylo možno dynamicky načítat a uvolňovat v průběhu aplikace.



Obrázek 6.1: Struktura hry

6.1.1 Skripty a GameObjekty

Hlavním skriptem hry je *GameController*. Jeho cílem je udržovat stav hry (připravena, pozastavena, ukončena nebo zrušena) a zaznamenávat hráčovo skóre s odehraným časem.

[1] http://www.blender.org/
[2] http://cgtextures.com/
[3] http://krita.org/

Výsledky po úspěšném dokončení hry předá instanci třídy *GameStatsManager*, který je implementován návrhovým vzorem jedináček. Jeho úkolem je ukládat a načítat hráčovi výsledky ze souboru a poskytovat je zbytku aplikace.

Pro dynamické menu, scén, labyrintů a pro zobrazení načítací obrazovky slouží skript *LevelsManager*. Ten po změně scény, zobrazí načítací obrazovku a po dvou sekundách spustí samotné načítání scény, To z důvodu, že ve verzi Unity Free nelze načítat scény v dalším vlákně[16]. Pokud se načetla herní scéna tak po dokončení načítání je jeho úkolem načíst ze složky Assets/Resources požadovaný prefab labyrintu.

Všechny labyrinty je možné hrát ve více verzích, které se liší počtem kuliček. Ty se podle vybrané verze vytváří ve skriptu *BallsManager*. Ten je nejprve vytvoří, a zařadí do fronty čekajících kuliček, ze které se postupně kuličky umisťují do scény. Pro každou kuličku se vygeneruje náhodná pozice, která vychází z pozice jednoho z množiny vstupních bodů. K té se připočítá náhodný ofset a na základě této pozice je kulička umístěna do scény.

Každá kulička obsahuje skript *BallController*, který kontroluje zda nevyskočila nebo nepropadla labyrintem dolů. Jestliže se tak stane, umístí ji do fronty čekajících kuliček k opětovnému umístění do labyrintu.

Všechny důlky obsahují imaginární kolizní objekt ve tvaru koule, na který je navázán skript *HoleController*. Dojde-li k průchodu kuličky s tímto objektem, je zavolána událost *OnTriggerEnter* a v ní dojde k rozhodnutí, zda je důlek cílový nebo ne. Také podle příznaku aktivity se buď kulička vymaže ze scény a je místo ní určena jiná aktivní kulička nebo je umístěna do fronty čekajících kuliček k umístěni do labyrintu.

Jak popisuje kapitola 4.1 tak ve scéně musí být každý skript součástí nějakého GameObjektu. Proto jsem zvolil způsob, že každý logický prvek hry bude jeden GameObjekt. Například skripty pro řízení hry, práci s kuličkami jsou jeden GameObjekt. Skripty pro LeapMotion jsou součástí dalšího GameObjektu. Skript pro správu úrovní je další GameObjekt atd.... Každý z těchto speciálních GameObjektů má svůj tag, přes který je možní získávat referenci na daný GameObjekty a z něj skripty které jsou jemu přiřazeny. Všechny významné tagy jsou k definovány jako statické konstantní řetězce ve speciální třídě *Tags*.

6.2 Ovládání

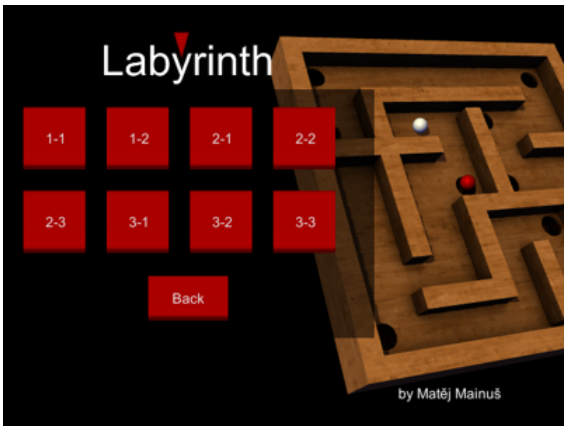
Naklánění herní desky podle údajů z knihovny poslané v kapitole 5.3 je realizováno třídou *LabController* a jejím potomkem *GameLabController*. Ta získá instanci třídy *LeapController* a z ní instanci *HandsBundle*. Jeho jednotkový normálový vektor podle citlivosti lineárně interpoluje s jednotkovým vektor orientovaným směrem vzhůru. Výsledný normálový vektor použije k výpočtu quaternionu, podle kterého se před každou simulací fyziky labyrint natočí.

6.3 Menu

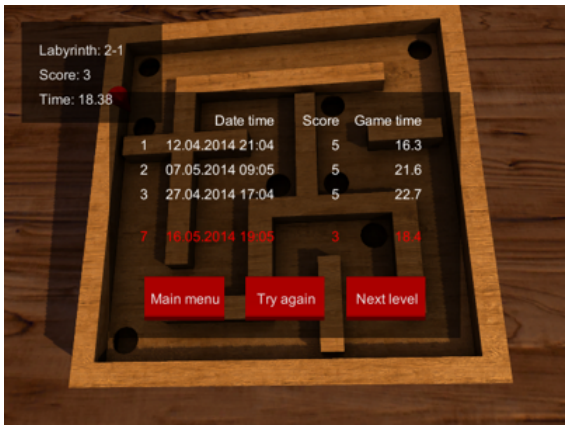
Pro vykreslení menu jsem použil GUI elementy, které jsou součástí Unity. Bloky s elementy v aplikaci pozicují tak, aby jejich poloha byla nezávislá na rozlišení obrazovky, elementy v rámci bloků se pozicují pomocí automatických rozložení. Některé prvky jako je například logo, aplikace vykresluje rozměrově nezávisle na rozlišení okna. Pro jednotnost rozměrů, jsem vytvořil třídu *GUIConsts*, která obsahuje definice velikosti rozměrů a mezer. Pro interakci s prvky menu pomocí kontroléru Leap Motion jsem využil vlastní knihovnu popsanou

v kapitole 5.3.3. Vykreslování hlavního menu je implementováno ve třídě *MainMenu*, jejíž hlavní částí je stavový automat reprezentující jednotlivé sekce menu.

Pro menu jsem vytvořil vlastní skin, ve kterém jsem upravil velikost písma, pozadí tlačítek, posuvníků a schránek pod bloky elementů.



(a) Menu s výběrem labyrintů.

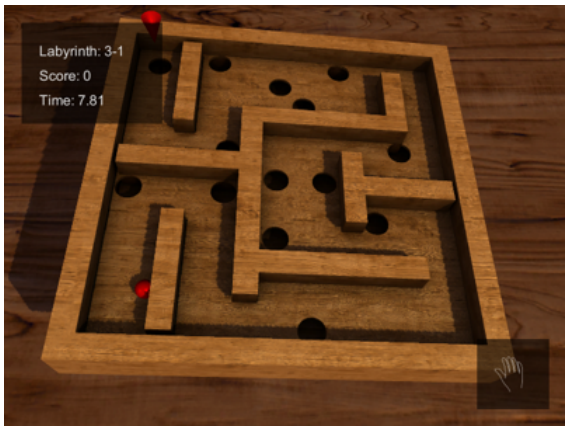


(b) Prezentace výsledků po dohrání hry.

6.4 Modely, textury, materiály, shadery, světla a animace

Do hry jsem vytvořil sadu labyrintů a modelů v modelovacím nástroji Blender. Vytvářel jsem je s důrazem na nízký počet polygonů, s důrazem na použití pouze jednoho materiálu na každý objekt, to z důvodu snížení vykreslovacích volání. Všechny materiály v projektu jsem vytvořil přímo v Unity a jsou na objekty aplikovány pomocí standardních shaderů dodávaných s enginem. Použil jsem hlavně shadery *Diffuse* a *Specular* a jejich *Bumpled* varianty[16]. Typ shaderu jsem volil podle vzdálenosti objektu od kamery, doby vyobrazení ve scéně a požadavku na vizuální dojem. Import objektů z Blendru se děje automaticky v Unity přes exportní formát FBX. Importované modely je nutné otočit kolem osy X o -90°, protože Blender používá jiný souřadný systém než Unity.

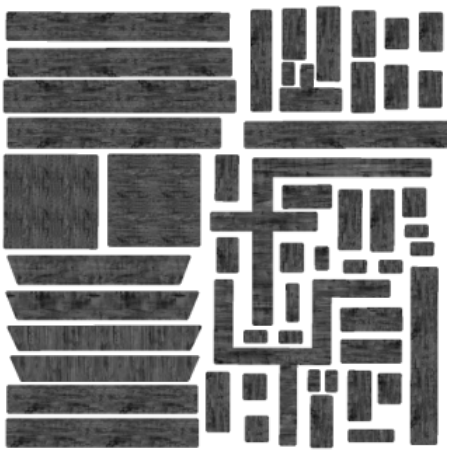
(a) Ukázka scény ze hry.



(b) Modelu labyrintu ze hry.

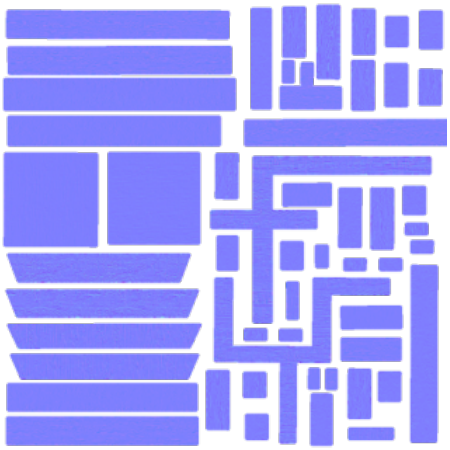
Materiály jsou ve většině případů složeny z hlavní textury a normálové mapy. Hlavní textura bývá často kombinace difuzní textury a textury ambientního osvětlení. Součástí většiny těchto textur je také odrazová textura která je v Unity umístěna do alfa kanálu hlavní textury. Na většinu modelů je použita technika UV mapování. Všechny textury mají rozlišení 1024x1024, ty které jsou dál od kamery nebo nejsou často ani vidět, jsou pomocí Unity zmenšeny aby se ušetřila paměť a výkon. Stažené textury jsem upravoval v programu Krita. Použil jsem funkce *Color-transfer* pro obarvení textury jinou texturou, funkce *Desaturate* a *Levels* pro převod obrázku do supně šedi. Také jsem upravoval některé textury na bezešvé[15].

(a) Difuzní textura s okolním osvětlením.



(b) Odrazová textura.

(c) Kombinace difuzní a odrazové textury.



(d) Normálová mapa labyrintu.

Obrázek 6.4: Ukázky textur labyrintu.

Nasvětlení scény je realizováno tmavě modrým ambientním osvětlením, jedním hlavními směrovým světlem světle žluté barvy, ze kterého se počítají stíny a sekundárním směrovým světlem tmavě oranžové barvy.

Animace kamery a šipky ukazující koncový důlek, jsem vytvořil přímo v Unity editoru, technikou klíčových snímků. Tyto animace jsou vytvořeny tak, aby se daly použít ve všech scénách a byly nezávislé na objektu se kterým animace pracuje.

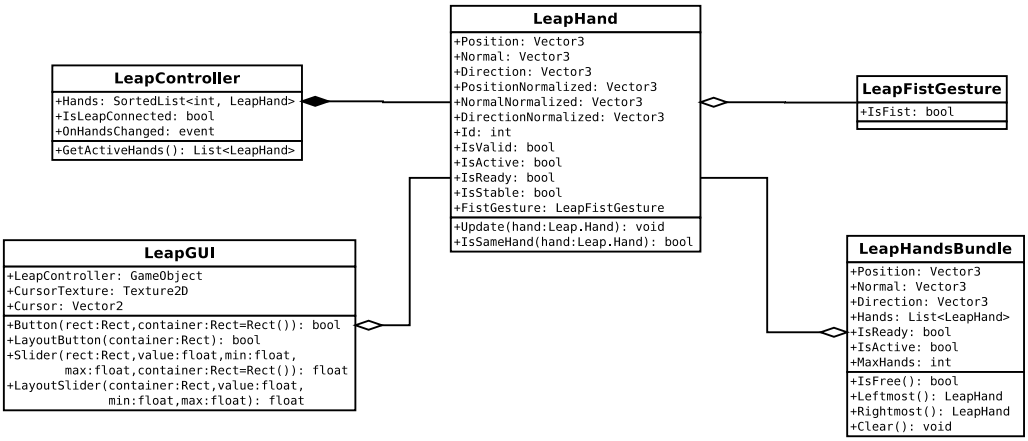
6.5 Knihovna rozšiřující Leap Motion SDK

Vstupním bodem je třída *LeapController*. Ta podle událostí v Unity zpracovává snímky z Leap Motion API. Podle detekovaných rukou vytváří instance třídy *LeapHand*, která má za cíl provázat ruku se stejným ID přes více snímků a uchovávat předchozí informace o ruce sloužící pro odhalení chyb způsobené špatnou detekcí. V této třídě jsou také implementovány algoritmy pro popsané v části 5.3.1.

Jelikož je ve hře podporován režim ovládání dvěma rukama, je součástí knihovny třída *LeapHandsBundle*, která dodává jednotné informace pro ovládání labyrintu nezávisle na tom, zda se ovládá jednou nebo dvěma rukama.

Součástí knihovny je také třída *LeapFistGesture*, jejíž úkolem je detekovat podle algoritmu v sekci 5.3.2 gesto sevření ruky v pěst.

Pro ovládání Unity GUI je v knihovně třída *LeapGUI*. Ta zajišťuje vykreslení kurzoru podle polohy ruky a poskytuje velmi podobné funkce pro práci s prvky GUI jako je tomu v Unity.



Obrázek 6.5: Diagram veřejného API knihovny.

Kapitola 7

Testování a hodnocení

V rámci testování jsem se snažil zjistit, jak rychle si uživatelé zvyknou na ovládání hry pomocí senzoru Leap Motion. Testování se zúčastnilo 12 lidí. Většina z nich neměla dosud žádnou zkušenost se zařízením Leap Motion. Proto jsem jim nejprve vysvětil princip ovládání a prakticky ukázal jak funguje. Také jsem je seznámil s cílem hry, pravidly a metrikami. Poté jsem je opakovaně nechal zahrát testovací labyrint, který má minimum překážek a je určený pro vyzkoušení ovládání. Výstupem toho testu je tabulka 7.1 průměrných časů jednotlivých pokusů, ve které jsou uživatelé rozděleni do dvou skupin podle předchozích zkušeností s kontrolérem Leap Motion.

Předchozí zkušenosti	Pokus 1	Pokus 2	Pokus 3
Začátečník	58s	34s	26s
Pokročilý	45s	29s	21s

Tabulka 7.1: Průměry časů ukazující rychlost osvojení ovládání jednoduchého labyrintu.

Cílem druhého testu bylo vysledovat učící křivku přesnosti ovládání, během hraní labyrintu 2-1. Metrikou testu byl čas dohrání a skóre jednotlivých pokusů, průměrné časy a skóre uživatelů jsou zapsány v tabulce 7.2

	Pokus 1	Pokus 2	Pokus 3	Pokus 4
Herní čas	74s	56s	42s	29s
Skóre	-8	-3	1	3

Tabulka 7.2: Průměrné časy a skóre ukazující učící křivku při hraní labyrintu 2-1

Při testování jsem vypozoroval následující věci. Skoro všichni uživatelé byli schopni velmi přesně ovládat labyrint už po druhé hře. První hra jim sice trvala v porovnání s ostatními velmi dlouho, ale od páté až šesté hry dále už nešlo vidět žádné rapidní zlepšení, protože v těchto hrách začala hrát roli i náhoda. Všichni uživatelé měli problém ovládat hru s větším množstvím kuliček. Soustředili se pouze na jednu což mělo za následek záporné skóre po dokončení hry. Jako důležitá vlastnost hry se ukázalo nastavení citlivosti. Hráči preferovali menší citlivost ovládání, proto jsem upravil jeho výchozí hodnotu. U uživatelů kteří dosud neměli zkušenost se senzorem Leap Motion jsem vypozoroval, že ze začátku pohybují rukama příliš blízko nebo naopak daleko od senzoru, který tak nemohl jejich pohyby dete-

kovat. Z testování také vyplynulo, že Leap Motion hůře rozpoznává orientaci levé než pravé ruky.

Výslednou hru jsem chtěl umístit do aplikačního obchodu Airspace, ve kterém se sdružují aplikace speciálně navržené nebo upravené pro ovládání s pomocí kontroléru Leap Motion. Před nahráním aplikace do obchodu je potřebné se zaregistrovat jako vývojář, vytvořit profil pro aplikaci, vyplnit potřebné údaje a poslat vytvořenou aplikaci do schvalovacího procesu. Během něj je aplikace velmi důkladně otestována lidmi z Leap Motion a výsledkem je velmi podrobný seznam chyb, problémů a doporučení, které se musí opravit. Pro zveřejnění mé aplikace, musím opravit pár drobností v menu, jako například přidat do hry tlačítko k vypnutí aplikace a vytvořit tutoriál. Také mi bylo doporučeno přidat více úrovní do hry, vylepšit ovládání menu a upravit tabulku výsledků.

Část připomínek z testování, jako je například vizuální odezva v menu, tlačítko ve hře k vypnutí aplikace a tutoriál jsem již zapracoval do aplikace, ale pro zveřejnění v aplikačním obchodě to není dostatečné. Také bych rád přidal do hry další mód ve kterém by se po umístění kuličky do koncového důlku náhodně vybral jiný důlek nebo aby pro každou kuličku byl předem určen jiný cílový důlek. Dále bych rád přidal animaci padající kuličky do důlku, rád bych obohatil hru o zvuky a přidal více netradičních úrovní například s šikmou podlahou.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat multiplatformní 3D hru labyrint, která k ovládání využije technologii Leap Motion. Výsledkem práce je aplikace složená z menu a samotné hry vytvořená v enginu Unity, která pro simulaci fyziky využívá engine PhysX a k sledování rukou Leap Motion SDK. Hra je od jiných ojedinělá svým ovládáním hlavně jeho přesností. Jejím jádrem je vlastní knihovna která integruje Leap Motion SDK do Unity, přidává podporu v ovládání uživatelského rozhraní, eliminuje chybně detekované objekty, řeší problém chybně přiřazených identifikačních čísel, dále eliminuje nedokonalosti v rozpoznávání orientace a přidává detekci gesta sevřeni ruky.

Aplikaci bych rád upravil, aby ji bylo možné umístit do obchodu AirSpace, obohatil ji o zvuky a přidal do ní nové režimy hraní. Vytvořil do ní nové netradiční labyrinty, například se šikmou podlahou nebo interaktivními prvky, které by více ztížili hraní.

Literatura

[1] Chen, F.-S., Fu, C.-M. a Huang, C.-L. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*. 2003, roč. 21, č. 8. S. 745–758.

[2] Clout, L. *Spate of injuries blamed on Nintendo Wii* [online]. 2008, 2008-01-22 [cit. 2. května 2014]. Dostupné z: http://www.telegraph.co.uk/news/uknews/1576244/Spate-of-injuries-blamed-on-Nintendo-Wii.html.

[3] Corporation, M. *Skeletal Tracking* [online]. [cit. 5. listopadu 2014]. Dostupné z: http://msdn.microsoft.com/en-us/library/hh973074.aspx.

[4] Johnson, S. *Daily Reacharound: Kinect Reactions* [online]. 2010, 2010-11-04 [cit. 2. května 2014]. Dostupné z: http://www.g4tv.com/thefeed/blog/post/708449/daily-reacharound-kinect-reactions/.

[5] Leap Motion, Inc. *API documentation* [online]. [cit. 20. dubna 2014]. Dostupné z: https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html.

[6] Leap Motion, Inc. *Leap Motion product* [online]. [cit. 2. května 2014]. Dostupné z: https://www.leapmotion.com/product.

[7] Manresa, C., Varona, J., Mas, R. et al. Hand tracking and gesture recognition for human-computer interaction. *Electronic letters on computer vision and image analysis*. 2005, roč. 5, č. 3. S. 96–104. ISSN 1577-5097.

[8] Nielsen, S. *When Bullets Move Too Fast* [online]. 2011, 2011-06-01 [cit. 25. dubna 2014]. Dostupné z: http://www.aorensoftware.com/blog/when-bullets-move-too-fast/.

[9] NVIDIA Corporation. *NVIDIA PhysX SDK Archives* [online]. [cit. 27. dubna 2014]. Dostupné z: http://www.nvidia.com/object/physx_archives.html.

[10] Potter, L. E., Araullo, J. a Carter, L. The Leap Motion Controller: A View on Sign Language. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*. New York, NY, USA: ACM, 2013. S. 175–178. ISBN 978-1-4503-2525-7.

[11] Reich, W. *C# Memory Management for Unity Developers* [online]. 2013, 2013-11-09 [cit. 4. května 2014]. Dostupné z: http://www.gamasutra.com/blogs/WendelinReich/20131109/203841/.

[12] Ren, Z., Meng, J., Yuan, J. et al. Robust Hand Gesture Recognition with Kinect Sensor. In *Proceedings of the 19th ACM International Conference on Multimedia*. New York, NY, USA: ACM, 2011. S. 759–760. MM '11. Dostupné z: http://doi.acm.org/10.1145/2072298.2072443. ISBN 978-1-4503-0616-4.

[13] Rios Soria, D. J., Schaeffer, S. E. a Garza Villarreal, S. E. Hand-gesture recognition using computer-vision techniques. 2013. S. 1–8.

[14] Schlömer, T., Poppinga, B., Henze, N. et al. Gesture Recognition with a Wii Controller. In *Proceedings of the 2Nd International Conference on Tangible and Embedded Interaction*. New York, NY, USA: ACM, 2008. S. 11–14. TEI '08. Dostupné z: http://doi.acm.org/10.1145/1347390.1347395. ISBN 978-1-60558-004-3.

[15] Tvrdý, L. *Create textures with Krita* [online]. 2013-03-16 [cit. 30. dubna 2014]. Dostupné z: http://lukast.mediablog.sk/log/?p=466.

[16] Unity Technologies. *Unity documentation* [online]. 2007-11-16 [cit. 27. dubna 2014]. Dostupné z: http://unity3d.com/learn/documentation.

[17] Weichert, F., Bachmann, D., Rudak, B. et al. Analysis of the accuracy and robustness of the leap motion controller. *Sensors (Basel, Switzerland)*. 2013, roč. 13, č. 5. S. 6380–6393. ISSN 1424-8220.

Příloha A

Obsah CD

- **src** - Projekt pro Unity se zdrojovými soubory aplikace
- **bin** - Přeložená aplikace pro platformy Windows x86_64 a OSX x86_64
- **doc** - Technická zpráva ve formátu PDF zdrojové texty pro prostředí LaTeX
- **video** - Prezentační video projektu a videa ukazující problémy zařízení Leap Motion

Příloha B

Manual

Tato část přílohy obsahuje informace o překladu aplikace a spuštění

B.0.1 Potřebný software

Aplikace pro překlad vyžaduje Unity verze 4.3.4 a Leap Motion SDK verze 1.0.9.8391. Pro svůj běh pak nainstalované ovladače pro zařízení Leap Motion

B.0.2 Překlad a spuštění

Pro přeložení aplikace je zapotřebí naimportovat složku *src* do editoru Unity, nakopírovat z Leap Motion SDK knihovny Leap.dll a LeapCSharp.dll do kořenové složky naimportovaného projektu a do podsložky Assets/Plugins knihovnu LeapCSharp.NET3.5.dll

Pro spuštění aplikace mimo editor je potřeba nakopírovat stejné soubory do kořenového adresáře s přeloženou aplikací.