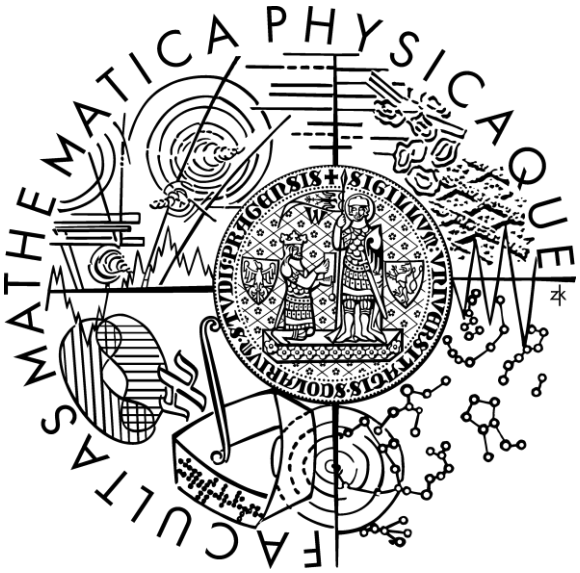
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Vladimír Mach

Atomix pro Microsoft Kinect pro Windows

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. David Hoksza, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové systémy

2014

Děkuji svému vedoucímu diplomové práce RNDr. Davidu Hokszovi, Ph.D. za konzultace a cenné rady během vedení diplomové práce. Dále bych také rád poděkoval své rodině a přátelům za jejich trpělivost a podporu během psaní této práce.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 29. 7. 2014 Vladimír Mach

Název práce: Atomix pro Microsoft Kinect pro Windows

Autor: Vladimír Mach

Katedra (ústav): Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. David Hoksza, Ph.D.

e-mail vedoucího: hokzsa@ksi.mff.cuni.cz

Abstrakt: Cílem diplomové práce je seznámit se se schopnostmi pohybového senzoru Kinect od společnosti Microsoft a následně implementovat hru Atomix, která pro své ovládání bude využívat přirozeného uživatelského rozhraní za využití senzoru Kinect. Práce popisuje vlastnosti senzoru Kinect a navrhuje vhodná gesta pro přirozené uživatelské rozhraní hry právě s ohledem na vlastnosti sensoru. Výsledná hra Atomix je implementována s pomocí herní knihovny XNA Framework. Kromě samotné hry je vytvořena aplikace pro vytváření vlastních herních úrovní. Práce také implementuje nezávislé knihovny obsahující třídy pro správu připojeného senzoru Kinect a komponenty pro XNA Framework využitelné i v jiných aplikacích a hrách.

Klíčová slova: atomix, kinect, xna

Title: Atomix for Microsoft Kinect for Windows

Author: Vladimír Mach

Department: Department of Software Engineering

Supervisor: RNDr. David Hoksza, Ph.D.

Supervisor's e-mail address: hokzsa@ksi.mff.cuni.cz

Abstract: The aim of this master thesis is to explore the capabilities of the motion sensing device Kinect by Microsoft and then implement Atomix game that will use natural user interface provided by the Kinect sensor. This thesis describes discovered capabilities and limitations of the sensor and proposes suitable gestures for natural user interface of the Atomix game with respect to the capabilities of the sensor. The resulting game is implemented using the XNA Framework library. Aside from the Atomix game is created an application for creating custom levels. Moreover this thesis also implements independent libraries containing classes for managing the connected Kinect sensors and components for XNA Framework that can be used in other application or games.

Keywords: atomix, kinect, xna

Obsah

Úvod

Předmětem této diplomové práce je naprogramovat hru *Atomix* pro současné počítače za využití moderních technologií – především pohybového senzoru *Microsoft Kinect*. Původní verzi hry *Atomix* vytvořila společnost Softtouch Productions v roce 1990 pro platformy Atari, Amiga a PC [1].

Senzor *Kinect* je zařízení od společnosti Microsoft, které je schopno detekovat pohyb uživatele v prostoru. Vznikl jako rozšíření herní konzole *XBOX 360* a poprvé byl veřejnosti ohlášen v červnu 2009 na veletrhu počítačových her Electronic Entertainment Expo 2009 [2].

K prvním zákazníkům se dostal po necelém roce (v listopadu 2010) a ihned se stal jedním z nejrychleji prodávaným zařízením. To dokazuje i fakt, že se po 60 dnech prodalo 8 milionů kusů senzoru a díky tomu získal rekord v Guinessově knize rekordů [2]. Záhy si senzor získal velkou oblibu i mezi vývojáři, kteří se snažili obejít omezení a zprovoznit senzor nejen na konzoli XBOX 360, ale i na klasických PC. Tuto snahu podpořila společnost Adafruit. Vyhlásila soutěž o 3 tisíce dolarů pro toho, kdo první vytvoří alternativní open source ovladač pro senzor *Kinect* funkční na PC, což se záhy podařilo a po 6 dnech od vyhlášení se podařilo alternativní ovladač zprovoznit[1].

Microsoft začátkem roku 2011 oznámil[2] vydání nekomerčního vývojového prostředí (SDK) cíleného především pro akademickou sféru a následně v květnu 2012 vydal[3] oficiální, komerčně použitelné SDK pro nepatrně upravený senzor s názvem *Kinect for Windows*.

Nová implementace hry využívá pohybového senzoru *Kinect* a hráčům umožňuje ovládání hry pohybem těla. Tato implementace zohledňuje i možnost využití hry na veřejně přístupných prostorech s častým střídání hráčů. Uživatelé mohou hru hrát, aniž by museli hledat myš nebo klávesnici, a k ovládání stačí pouze se přiblížit k senzoru a pohybovat rukou v prostoru.

[1] http://www.adafruit.com/blog/2010/11/10/we-have-a-winner-open-kinect-drivers-released-winner-will-use-3k-for-more-hacking-plus-an-additional-2k-goes-to-the-eff/ (navštíveno 7. 7. 2014)
[2] http://blogs.technet.com/b/microsoft_blog/archive/2011/02/21/kinect-for-windows-sdk-to-arrive-spring-2011.aspx (navštíveno 7. 7. 2014)
[3] http://blogs.msdn.com/b/kinectforwindows/archive/2012/05/21/kinect-for-windows-runtime-and-sdk-version-1-5-released.aspx (navštíveno 7. 7. 2014)

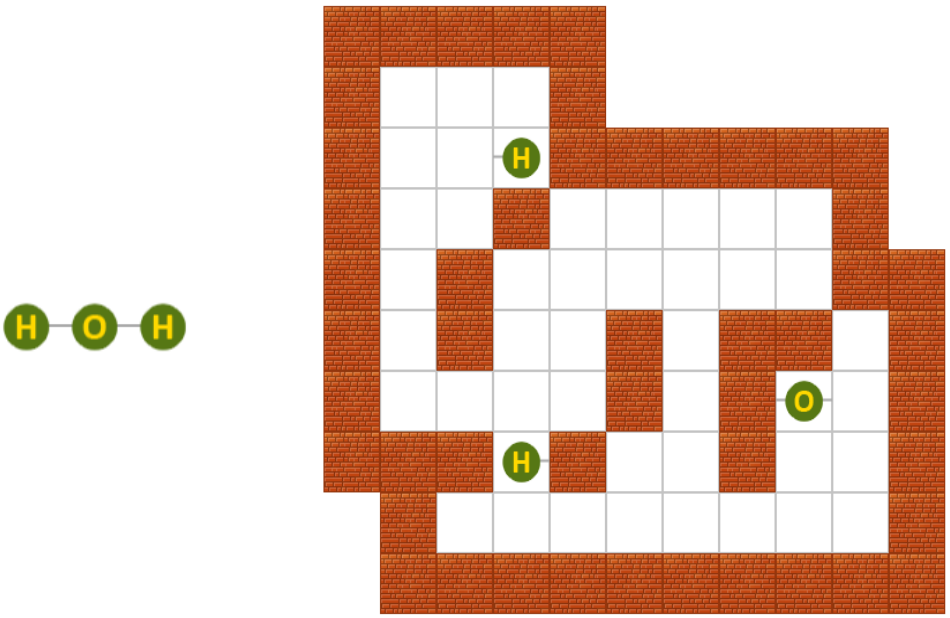
První kapitola této práce popisuje vlastnosti původní hry *Atomix*. Následující kapitola popisuje vlastnosti pohybového senzoru *Kinect* a dostupné vývojové prostředí pro práci s pohybovým senzorem. Ve třetí kapitole je navrženo přirozené uživatelské rozhraní pro novou implementaci hry s ohledem na zjištěné vlastnosti a omezení pohybového senzoru. Čtvrtá kapitola popisuje celkové členění projektu a technologie zvolené pro implementaci. Další kapitola popisuje samotnou implementaci obecných knihoven pro pohybový senzor *Kinect* a platformu XNA Framework. Šestá kapitola se věnuje implementaci samotné hry a obslužných aplikací. Součástí práce je také uživatelská dokumentace hry v příloze C a uživatelská dokumentace aplikace pro zaznamenání gest v příloze D.

1. Hra Atomix

Cílem hry *Atomix* je složit předem známé molekuly z různě umístěných atomů po hrací ploše. V každé herní úrovni je zadána jiná molekula.

Hrací úroveň se skládá z předepsané molekuly a hrací plochy. Hrací plocha hry je konečná dvourozměrná mřížka dlaždic různého typu. Ve hře jsou rozlišovány tři typy dlaždic – zeď, atom nebo volný prostor. Zdí nelze po hrací ploše pohybovat. Atomy jsou různého druhu, liší se typem atomu a vazbou. Po hrací ploše jimi lze pohybovat pouze vertikálně nebo horizontálně. A to pouze k nejbližší překážce v daném směru, nemůžou se zastavit volně v prostoru. Za překážku se považuje buď zeď nebo jiný atom [1]. Atom smí být umístěn pouze na dlaždicích s volným prostorem.

Ukázka jedné úrovně hry se zadanou molekulou a hrací mřížkou pro tuto molekulu je na obrázku 1.



(a) (b)

Obrázek 1: Úroveň hry: (a) zadaná molekula; (b) hrací plocha.

Úroveň je vyhrána, pokud jsou poskládány jednotlivé atomy po hrací ploše do molekuly, která byla zadána. Takovýchto řešení může existovat více, liší se například počtem nutných tahů pro složení hry.

Původní hra počet tahů nepočítala, pouze měla časový limit, ve kterém bylo nutné molekulu složit. Implementace v této práci místo pevného časového limitu počítá jednotlivé tahy potřebné ke složení molekuly.

Za lepší řešení se považuje takové, u kterého je potřeba méně tahů ke složení. Pokud je počet tahů stejný, jako u už existujícího nejlepšího řešení, pak se porovná čas.

Od doby vydání původní hry vznikla spousta alternativních implementací, například v jazyce JavaScript pro webový prohlížeč[4] nebo v GTK pro grafické rozhraní GNOME[5].

[4] http://kp-atomix.googlecode.com/hg/index.html (navštíveno 28. 7. 2014)
[5] https://github.com/GNOME/atomix (navštíveno 28. 7. 2014)

2. Senzor Kinect

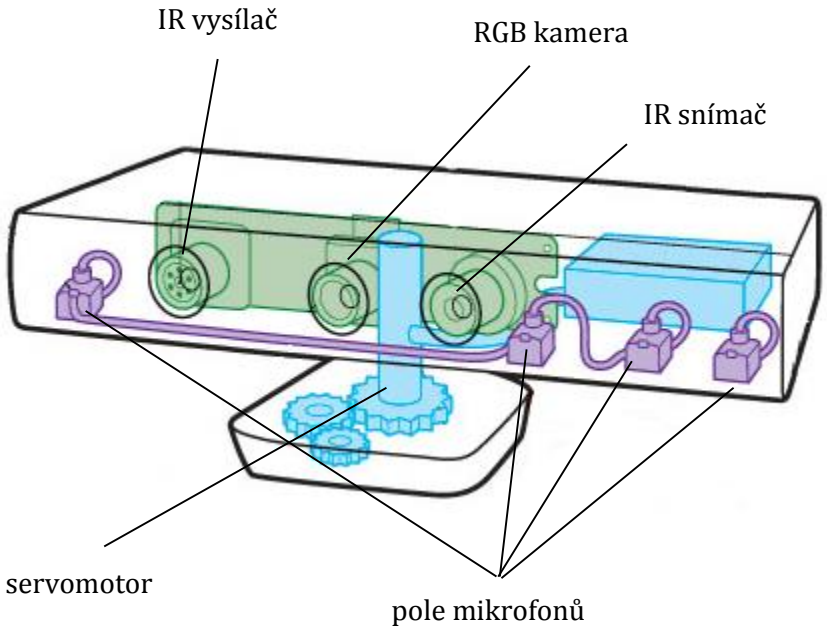
Tato kapitola popisuje hardwarové vlastnosti senzoru *Kinect* a dostupné vývojové prostředí pro použití senzoru ve vlastních aplikacích.

2.1. Hardware

Senzor *Kinect* se skládá z barevné kamery, infračerveného vysílače, infračervené kamery, pole mikrofonů, akcelerometru a servomotoru. Rozmístění jednotlivých komponent je znázorněno na obrázku 2. Za pomoci infračerveného spektra senzor snímá vzdálenost překážek od senzoru.

Pole mikrofonu je složeno ze čtyř mikrofonů, pomocí kterých je možné rozlišit směr zaznamenaného zvuku a lze tak odhadnout který hráč promluvil, tzv. *beamforming*. Mikrofon je schopen zaznamenat mluvené slovo i ze tří metrů od senzoru, podporuje potlačení šumu a ozvěny [3].

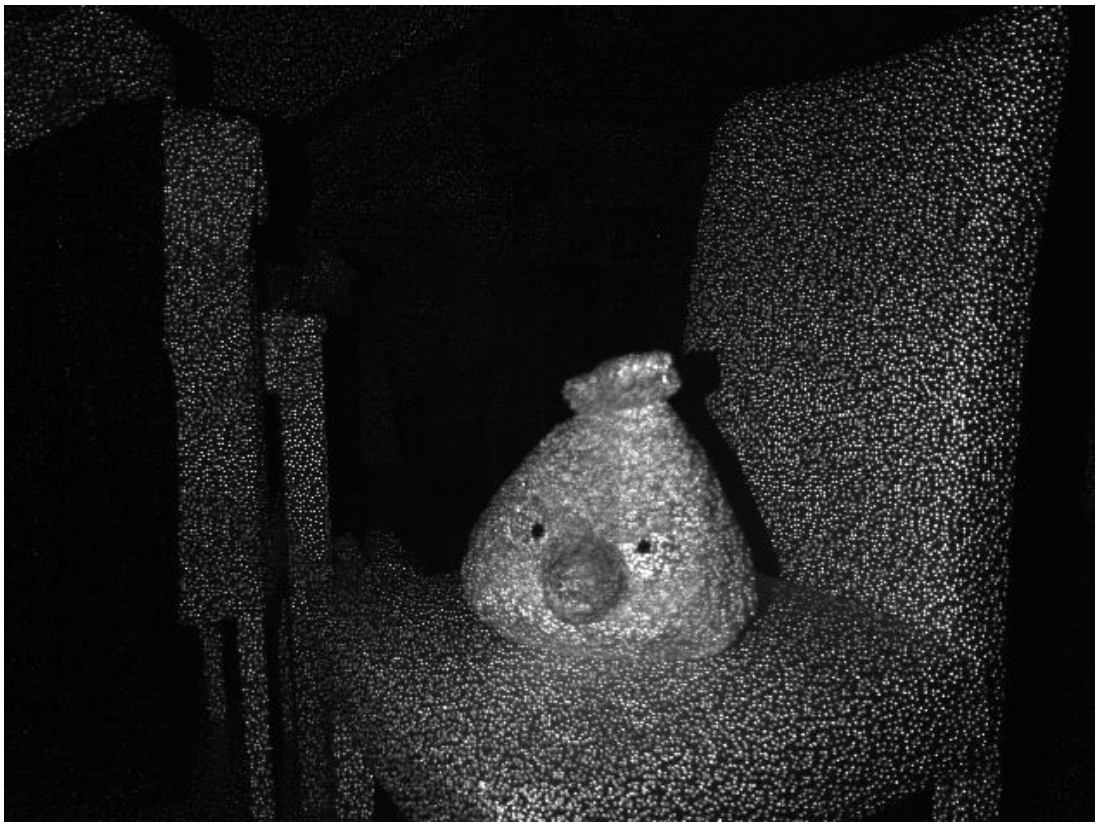
Pomocí akcelerometru je senzor schopen rozpoznat i neobvyklé situace jako například otočení o 180°, díky čemuž by mělo být možné senzor připevnit i na strop.



Obrázek 2: Komponenty senzoru Kinect. Obrázek převzat z [4].

Senzor měří vzdálenost tak, že před sebe infračerveným vysílačem vysílá předem známou mřížku bodů. Scénu ozářenou touto mřížkou následně pomocí infračerveného snímače snímá a porovnává s referenční mřížkou. Referenční mřížka je získána zaznamenáním roviny v předem známé vzdálenosti od senzoru a poté je uložena

v paměti senzoru [5]. Vzdálenost bodů je získána pomocí triangulace mezi referenční a zaznamenanou mřížkou [6]. Ukázka vysílané mřížky infračervenou kamerou je na obrázku 3.



Obrázek 3: Mřížka zaznamenaná infračervenou kamerou.

Senzor se k počítači připojuje přes klasický USB port. V dokumentaci je zmíněno omezení, že senzor *Kinect* by měl být připojen jako jediné zařízení na daném USB řadiči. V praxi senzor *Kinect* funguje, i když je na stejný USB řadič připojena například klávesnice. Ale pokud by se připojily na stejný USB řadič senzory dva, tak v jednu chvíli je možné mít funkční pouze jeden a druhý už spustit nelze. Senzor *Kinect* se následně ve správci zařízení tváří jako složené zařízení.

Pokud senzor *Kinect* zjistí, že USB řadič není schopen poskytnout požadovanou kapacitu pro plynulé fungování, vrátí jako návratový status `InsufficientBandwidth` během inicializace. Senzor také kontroluje, zda je připojeno dodatečné externí napájení, které je nutné pro fungování všech kamer. Pokud napájení není připojeno, tak vrátí status `NotPowered`.

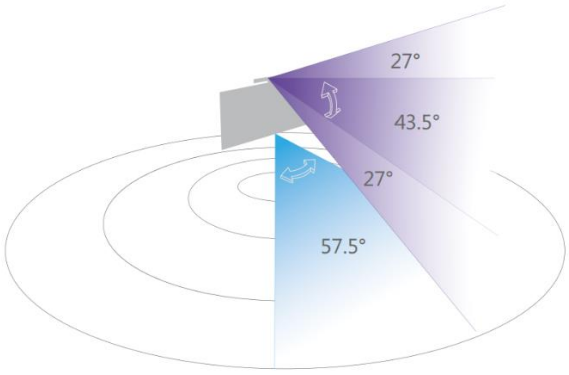
V současné době existují dvě varianty senzoru *Kinect* – *Kinect for XBOX 360* a *Kinect for Windows*. Pro vývoj a komerční použití je oficiálně podporován pouze ve verzi *Kinect for Windows*. Největším rozdílem mezi nimi je to, že firmware verze pro *Windows* má tzv.

blízký režim, který umožňuje snímání uživatele při kratší vzdálenosti od senzoru. Motivací pro tuto změnu bylo umožnit snímat uživatele sedícího u počítače. Tento režim snímání lze aktivovat při inicializaci senzoru. Přestože oficiálně je společností Microsoft k vývoji podporován pouze *Kinect for Windows*, je pomocí dodatečného adaptéru možné připojit k počítači i senzor *Kinect* od herní konzole XBOX 360, a to se stejnými ovladači, jako v případě verze pro *Windows*. Při každém spuštění aplikace používající *Kinect* se však vypíše upozornění, že toto není podporovaný scénář:

```
The Kinect sensor plugged into your computer is for use on the Xbox 360.
You may continue using your Xbox 360 Kinect sensor on your computer for development
purposes.
```

Zároveň je *Kinect for XBOX 360* funkční pouze, pokud je kromě základního běhového prostředí (Kinect for Windows Runtime) také nainstalováno kompletní vývojové prostředí (Kinect for Windows SDK). Vývojové prostředí je určeno pro instalaci na vývojářské počítače a není zamýšleno k distribuci a instalaci na počítače běžných uživatelů.

Pozorovací úhly [6] senzoru jsou 57° vodorovně a 43° svisle, tento rozsah lze ještě naklopením senzoru pomocí servomotoru upravit v rozsahu ∓27°. Naklopení lze ovládat pouze softwarově pomocí SDK, uživatel by neměl mechanicky senzorem naklápět. Proto je důležité, aby aplikace při inicializaci senzoru nastavila odpovídající úhel nebo uživateli poskytla rozhraní pro změnu orientace. Při nastavování úhlu naklopení se využívá hodnot z akcelerometru, čímž se srovnává nepřesnost podkladu, na kterém je pohybový senzor postaven.



Obrázek 4: Snímací úhly senzoru. Obrázek převzat z [4].

[6] http://msdn.microsoft.com/en-us/library/jj131033.aspx (navštíveno 10. 7. 2014)

Optimální rozsah vzdálenosti uživatele od senzoru je od 120 centimetrů do 4 metrů pro běžný režim případně 40 centimetrů do 3 metrů pro režim blízko senzoru. Režim snímání blízko senzoru je podporován pouze verzí *Kinect for Windows*.

2.2. SDK

Kromě oficiálního SDK [7] od firmy Microsoft existují i alternativní SDK, například OpenKinect[8] nebo OpenNI[9]. Oficiální SDK podporuje vývoj jak nativních aplikací v jazyce C++, tak i aplikací pro platformu .NET Framework. Alternativní SDK umožňují používání senzoru *Kinect* i na jiných platformách než Windows.

Tato práce využívá pouze variantu SDK pro platformu .NET oficiálního SDK od společnosti Microsoft, a to ve verzi 1.8. Proto se zbytek textu práce bude věnovat pouze funkcionalitě oficiálního SDK.

Připojený senzor Kinect je v SDK reprezentován třídou `KinectSensor`, která zprostředkovává přístup k těmto datům ze senzoru:

Audio Stream zvuk zaznamenaný senzorem; kapitola 2.2.1

Color Stream video zaznamenané senzorem; kapitola 2.2.2

Depth Stream zjištěné vzdálenosti od senzoru; kapitola 2.2.3

Skeleton Stream senzorem rozpoznané postavy; kapitola 2.2.4

Kromě zvuku, jehož zpracování se věnuje kapitola 2.2.1, se poskytovaná data mohou získávat obsluhou událostí, což je ideální pro klasické aplikace, nebo pomocí explicitního dotazování, tzv. *polling*, které je vhodnější u her pro zpracování dat během herní smyčky.

Následující kapitoly se podrobněji věnují jednotlivým typům dat, které senzor poskytuje.

2.2.1. Audio Stream

Zvuk lze se senzoru získat pomocí vlastnosti `AudioSource` na třídě `KinectSensor`, která vrací instanci třídy `KinectAudioSource`.

Třída `KinectAudioSource` slouží jako obálka poskytující zjednodušený přístup k nativní implementaci KinectAudio DMO (DirectX Media Object), která rozšiřuje základní DMO systému Windows o podporu odstranění šumu, a detekci směru odkud zvuk přichází, tzv. *beamforming*.

[7] http://www.microsoft.com/en-us/download/details.aspx?id=40278 (navštíveno 10. 7. 2014)
[8] http://openkinect.org/wiki/Main_Page (navštíveno 10. 7. 2014)
[9] http://structure.io/openni (navštíveno 10. 7. 2014)

2.2.2. Color Stream

Kinect podporuje různé formáty – RGB, YUV, Bayer – lišící se podporovaným rozlišením a počtem snímků za vteřinu. Nejvyšší senzorem podporované rozlišení je 640×480 pixelů při 30 snímcích za vteřinu. Podrobnosti o dostupných formátech a jejich rozlišeních jsou rozepsány v oficiální dokumentaci [3]. Senzor umožňuje aktivně zpracovávat pouze jeden typ videa. Pokus o otevření druhého typu videa způsobí, že předchozí typ se zastaví a nahradí se novým.

Data lze získat z události `ColorFrameReady` nebo explicitně zavoláním metody `OpenNextFrame` na vlastnosti `ColorStream` třídy `KinectSensor`. V obou případech je získána instance třídy `ColorImageFrame` obsahující metodu `CopyPixelDataTo`, která vrací pole typu `byte`. Délku vráceného pole lze zjistit z nastaveného formátu dat, případně každý vrácený snímek obsahuje údaj o délce pole ve vlastnosti `PixelDataLength`. Počet za sebou jdoucích bytů na jeden pixel lze zjistit z vlastnosti `BytesPerPixel`.

Kromě barevného videa lze stejným způsobem ze senzoru získat obraz z infračervené kamery. Pro získání výstupu z infračervené kamery je nutné vlastnost pro formát obrazu `ColorImageFormat` nastavit na hodnotu `InfraredResolution640x480Fps30`.

Jak povolit výstup barevného videa ze senzoru a jeho následné získání je vidět na ukázce 1.

```
private void EnableVideoStream(KinectSensor sensor)
{
   sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
   sensor.ColorFrameReady += ColorFrameReady;
}

private void ColorFrameReady(object s, ColorImageFrameReadyEventArgs e)
{
   using (ColorImageFrame frame = e.OpenColorImageFrame())
   {
       // načtení dat ze snímku
       byte[] pixelData = new byte[frame.PixelDataLength];
       frame.CopyPixelDataTo(pixelData);
   }
}
```

Ukázka 1: Povolení videa ze senzoru Kinect a jeho následné přečtení.

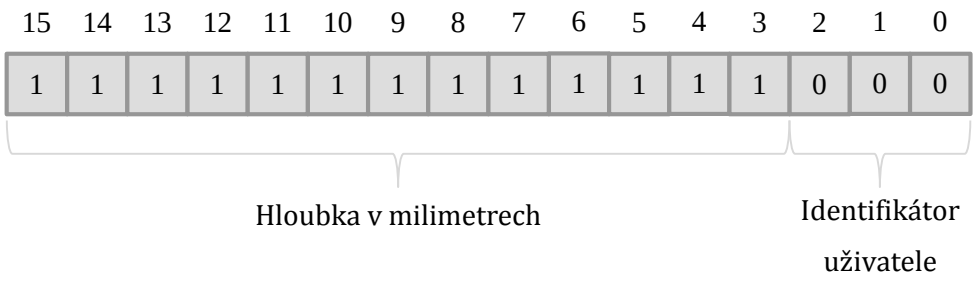
2.2.3. Depth Stream

Depth Stream zprostředkovává vzdálenosti objektů od senzoru *Kinect*. Data o vzdálenostech jsou předávána vždy s rychlostí 30 snímků za vteřinu a to v rozlišeních

80×60, 320×240 nebo 640×480 pixelů. Pro každý pixel je uvedena vzdálenost k nejbližší překážce od senzoru a identifikátor detekované postavy [3].

Data ze senzoru lze získat z události `DepthFrameReady` nebo explicitně zavoláním metody `OpenNextFrame` na vlastnosti `DepthStream` třídy `KinectSensor`. V obou případech je získána instance třídy `DepthImageFrame` obsahující metodu `CopyPixelDataTo`, která vrací pole typu `short` po jednom prvku pro každý pixel. Délku vráceného pole lze zjistit z nastaveného formátu dat, případně každý vrácený snímek obsahuje údaj o délce pole ve vlastnosti `PixelDataLength`.

Každý pixel je reprezentován typem `short`, který má délku 16 bitů. Prvních 13 nejvýznamnějších bitů obsahuje údaj o vzdálenosti, zbylé tři nejnižší bity obsahují identifikátor postavy. Toto rozdělení je znázorněno na obrázku 5.



Obrázek 5: Rozdělení hodnot pro každý pixel vrácených dat.

K oddělení těchto dvou hodnot lze využít konstant `PlayerIndexBitmaskWidth` a `PlayerIndexBitmask` definovaných na třídě `DepthImageFrame`.

Příklad, jak získat data o hloubce ze senzoru a zjistit hloubku a identifikátor postavy ze senzoru *Kinect*, je v ukázce 2.

```
using (DepthImageFrame depthFrame = sensor.DepthStream.OpenNextFrame(0))
{
   short[] pixelData = new short[depthFrame.PixelDataLength];
   depthFrame.CopyPixelDataTo(pixelData);
   int index = 0;

   int playerId = pixelData[index] & DepthImageFrame.PlayerIndexBitmask;
   int depth = pixelData[index] >> DepthImageFrame.PlayerIndexBitmaskWidth;
}
```

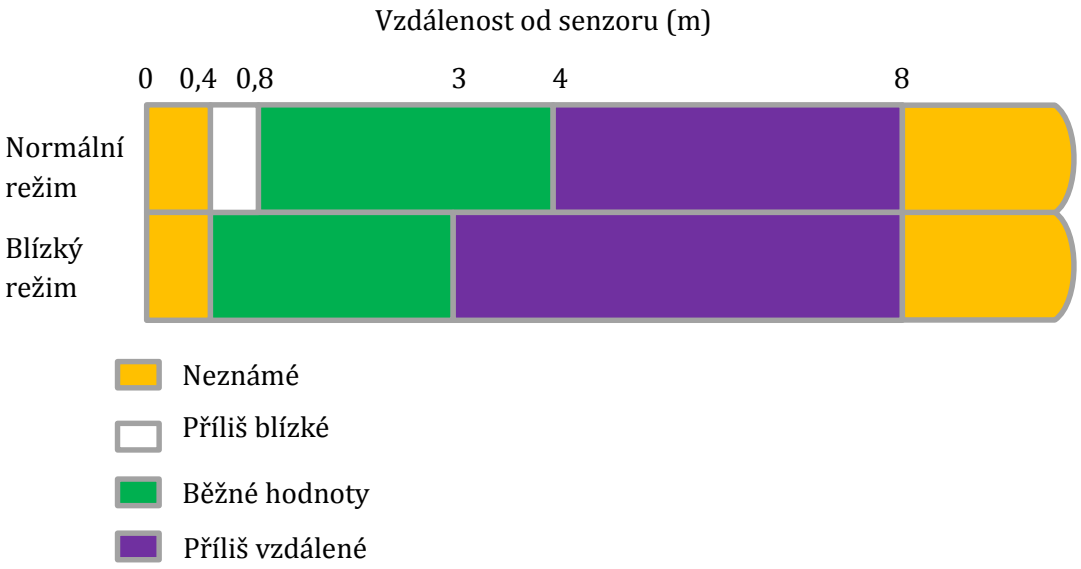
Ukázka 2: Načtení dat z Depth Stream a zjištění identifikátoru postavy a hloubky vybraného pixelu.

Vzdálenosti, které senzor vrací, jsou v milimetrech, a kromě běžných hodnot vrací také speciální konstanty, které jsou rozepsány v tabulce 1.

Hodnota	Význam
`0x0000`	objekt je příliš blízko senzoru
`0x0FFF`	objekt je příliš vzdálený od senzoru
`0x1FFF`	nepodařilo se zjistit vzdálenost

Tabulka 1: Speciální hodnoty pro hloubku.

Rozsah měřených vzdáleností se liší podle nastaveného režimu senzoru, jak je znázorněno na obrázku 6.



Obrázek 6: Rozsah měřitelných vzdáleností.

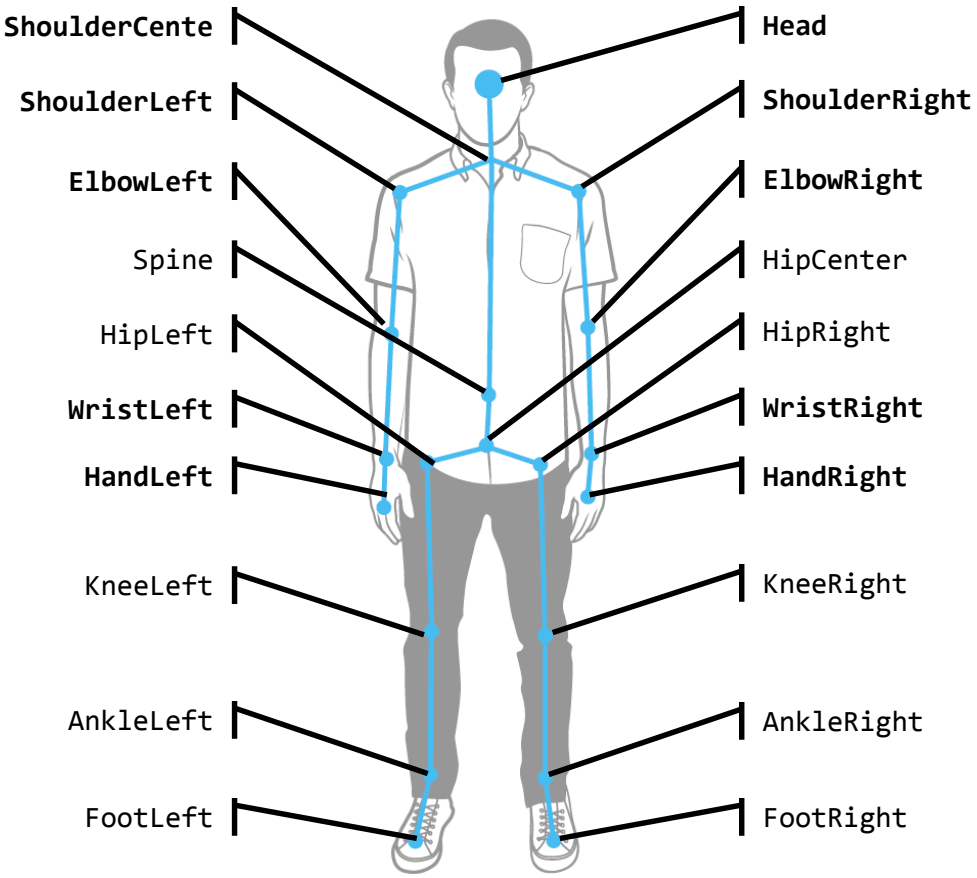
Se zvyšující se vzdáleností překážky od senzoru roste nepřesnost naměřených hodnot. Při vzdálenosti 3 metrů od senzoru je tolerance naměřených bodů 2,5 centimetru [7].

Pokud žádný uživatel na dané souřadnici nebyl detekován, je vrácena hodnota `0`. Senzor zvládá rozlišit až 6 postav. Identifikátor postavy získaný z údajů o hloubce lze následně použít pro získání podrobností o detekované kostře.

2.2.4. Skeleton Stream

Kinect dokáže rozpoznat až 6 postav ve svém zorném poli, ale pouze u dvou postav dokáže detailněji sledovat jejich pohyb. Rozpoznávání postav je optimalizováno na postavy stojící nebo sedící čelem k senzoru. K rozpoznávání postavy stačí, aby

v dosahu senzoru byla viditelná hlava a horní část těla [8]. Pro sledování více jak dvou postav souběžně by bylo nutné připojit další senzor Kinect. Ovšem díky způsobu, jakým senzor měří hloubku v prostoru (viz kapitola 2.1), by se vyzařované infračervené mřížky navzájem překrývaly a to by zapříčinilo vzájemné rušení a nepřesnosti v měření hloubky. Senzor rozlišuje při aktivním sledování postavy 20 bodů na těle, které jsou vyznačeny na obrázku 7.



Obrázek 7: Senzorem rozpoznávané body na postavě; popisky bodů odpovídají hodnotám ve výčtovém typu JointType. Obrázek převzat z [4].

U sledování postavy lze přepínat mezi sledováním celé nebo sedící postavy. První režim sleduje všech 20 bodů na postavě. Sedící režim sleduje jen horní končetiny, hlavu a střed ramen, tyto body jsou na obrázku 7 označeny tučně.

Postava je sledována aktivně, pokud ji senzor sleduje celou, tj. všech 20 známých bodů.

Postava je sledována pasivně, pokud senzor sleduje pouze její pozici v prostoru.

Data o sledovaných postavách jsou vraceny v objektu typu `SkeletonFrame`, který obsahuje unikátní číslo vráceného rámce, pole detekovaných postav, délku tohoto pole

a odhad roviny, ve které se osoba nachází. Tento vektor lze využít například pro odstranění pozadí nebo rozlišení více postav od sebe.

Díky nepřesnému výstupu z infračerveného senzoru se pro zjišťování bodů těla provádí korekce výstupu, kterou lze ovlivnit pomocí struktury `TransformSmoothParameters` předané při inicializaci *Skeleton stream*. Přehled nastavitelných parametrů pro vyhlazování je uveden v tabulce 2. Podrobnosti o parametrech korekce lze nalézt v dokumentaci [9].

Parametr	**Význam**
`Smoothing`	Parametr určuje úroveň vyhlazování bodů postavy mezi jednotlivými voláními. Čím je vyšší úroveň vyhlazování, tím je větší latence mezi. Při hodnotě 0 jsou vrácena původní neupravená data.
`Correction`	Parametr určuje úroveň korekce původních dat. Čím nižší hodnota, tím je korekce přesnější, ale pomalejší.
`Prediction`	Parametr určuje množství snímků, které senzor bude předpovídat.
`JitterRadius`	Parametr určuje poloměr korekce v metrech.
`MaxDeviationRadius`	Parametr určuje poloměr v metrech, o kolik se musí lišit souřadnice bodu od předchozích, aby se už považovaly za novou pozici bodu.

Tabulka 2: Přehled parametrů pro vyhlazování.

Příklad nastavení vlastních hodnot korekce je v ukázce 3.

```
TransformSmoothParameters parameters = new TransformSmoothParameters();
parameters.Smoothing = 0.5f;
parameters.Correction = 0.1f;
parameters.Prediction = 0.5f;
parameters.JitterRadius = 0.1f;
parameters.MaxDeviationRadius = 0.1f;

sensor.SkeletonStream.Enable(parameters);
```

Ukázka 3: Vlastní nastavení vyhlazování.

3. Návrh uživatelského rozhraní

Při navrhování uživatelského rozhraní pro hru *Atomix* bylo cílem využít přirozeného uživatelské prostředí (NUI), kterého je možné docílit použitím pohybového senzoru *Kinect*. NUI je definováno tak, že není potřeba žádného dalšího zařízení, například klávesnice, pro interakci s počítačem. Při návrhu uživatelského rozhraní jsou brány v potaz technické limity pohybového senzoru zmíněné v kapitole 2.

V původní verzi hry existovala varianta pro dva hráče, ve které měl každý hráč pevně daných 30 sekund pro provedení tahu na stejné hrací ploše [1]. V této práci je implementována pouze verze s podporou pro jednoho hráče.

3.1. Detekce hráče

Senzor *Kinect* je schopen v jednu chvíli detekovat více než jednu postavu, podrobněji viz kapitola 2.2.4. Proto je nutné zajistit vhodnou detekci aktuálního hráče, aby například náhodný kolemjdoucí nepřevzal kontrolu nad hrou aktuálně hrajícího uživatele. Nabízí se několik strategií, jak hráče z dat získaných ze senzoru vybrat.

SDK vrací jak aktivně sledované postavy s pozicemi všech 20 sledovaných bodů, tak i pasivně sledované postavy, u kterých je známá jen orientační pozice v prostoru. Způsob zjištění této pozice není v dokumentaci [3] specifikován. Pro použití ve hře je třeba omezit výběr postav získaných ze senzoru jen na takové postavy, které jsou sledovány aktivně, jelikož pro rozpoznávání gest je nutné znát pozice jednotlivých sledovaných bodů.

Následující tři kapitoly popisují různé strategie pro výběr sledované postavy hráče.

3.1.1. První sledovaná osoba

Seznam rozpoznaných postav je ze senzoru předáván jako pole. Nejsnazší strategie pro výběr sledované osoby je použít první postavu ze získaného pole, která je senzorem sledována. SDK ovšem negarantuje pořadí, v jakém jsou postavy v poli řazeny mezi jednotlivými voláními. Proto není jisté, zda takto získaná osoba bude v následujícím poli zachována na stejném indexu.

3.1.2. Pevně zvolená osoba

Senzor každé sledované postavě přiřazuje unikátní číselný identifikátor. Pomocí tohoto identifikátoru lze postavu následně rozlišit v datech vrácených ze sensoru. Tento identifikátor je unikátní pouze po dobu, kdy je postava senzorem sledována.

Získat identifikátor postavy hráče, který chce ovládat hru, lze například tak, že hra vyzve hráče k provedení rozpoznávacího gesta. Hra by si poté pamatovala identifikátor postavy, která provedla rozpoznávací gesto a pouze tato postava by směla ovládat hru.

Pokud hráč odejde z oblasti snímané senzorem a následně se vrátí zpět, není garantováno, že senzor postavě stejného hráče přidělí původní identifikátor. Z pohledu senzoru se bude jednat o novou postavu nezávislou na předchozí sledované.

Pokud tedy hráč na krátký odejde z dosahu senzoru, hra bude pozastavena a pro pokračování hry bude nutné provést rozpoznávací gesto znovu.

3.1.3. Nejbližší sledovaná osoba

Senzor kromě pozic jednotlivých sledovaných bodů na těle vrací i pozici pro celou postavu. Tuto pozici senzor vrací i pro postavy, které nejsou sledovány aktivně.

Tuto pozici lze využít pro určení nejbližší postavy od senzoru – ze všech získaných postav se vybere ta, která má nejmenší hodnotu souřadnice v ose z.

Tato strategie se jeví jako nejpřirozenější pro hráče. Pro vystřídání stačí, aby se hráč, který chce ovládat hru, postavil blíže k senzoru.

3.2. Kurzor

Kurzor je důležitou součástí pro navigaci po hrací ploše. Pro určení pozice kurzoru je využito sledování pohybu dlaně. Ne každému uživateli může vyhovovat sledování předem dané dlaně, proto je umožněno uživateli si zvolit, zda chce pro pohyb kurzoru využít pravou nebo levou dlaň. Toto rozlišení sledované dlaně je možné zařídit buď přes pevně zapamatované nastavení, nebo dynamicky během používání. Dynamickou volbu dlaně lze realizovat například sledováním té dlaně, která je vzdálenější od zbytku těla. Pro změnu dlaně tedy stačí prohodit ruce.

Na určení pozice kurzoru je vhodné nastavit určitou minimální vzdálenost dlaně od těla, při které bude mapována na kurzor. Snahou tohoto omezení je zabránit nechtěným posunům kurzoru během provádění gest.

Samotné mapování pozice dlaně v prostoru na plochu obrazovky lze provést absolutně nebo relativně.

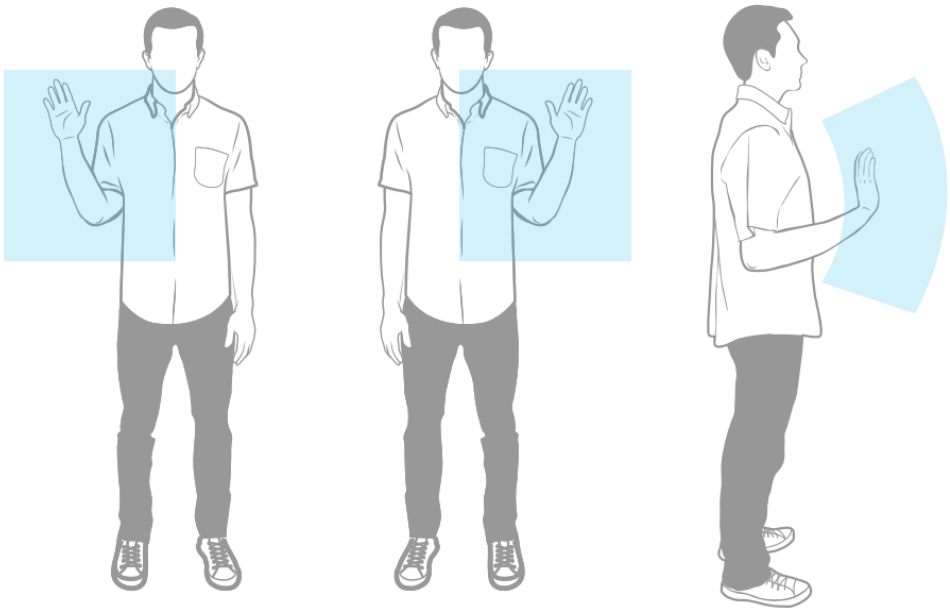
3.2.1. Absolutní mapování kurzoru

Absolutní mapování kurzoru je nejjednodušší způsob, jakým lze mapovat pozici dlaně na kurzor. Z dat ze senzoru je získána reálná pozice dlaně v prostoru. Tato pozice se následně přepočítá na osové souřadnice pixelu, který v RGB videu odpovídá této reálné

pozici. Poté se vypočítá poměr souřadnic tohoto pixelu vůči rozměrům RGB videa a ve stejném poměru se umístí kurzor na obrazovce.

Absolutní mapování vyžaduje, aby se uživatel posunul celým tělem po senzorem snímané ploše. Pokud je například uživatel ve větší vzdálenosti od senzoru, a chce posunout kurzor z levého horního rohu do pravého horního rohu, musí udělat několik kroků doprava, aby posun kurzoru byl dostatečně velký.

Tento způsob pohybu není pro uživatele přirozený a ani neodpovídá optimální interakční zóně (PHIZ) znázorněné na obrázku 8 [4].



Obrázek 8: Optimální interakční zóna (PHIZ). Obrázek převzat z [4].

3.2.2. Relativní mapování kurzoru

Pro respektování optimální interakční zóny uživatele je nutné kurzor na obrazovce mapovat relativně vůči postavě uživatele ovládajícího kurzor při zachování optimální interakční zóny. Plocha optimální interakční zóny je relativní vůči výšce uživatele a jeho pozici v prostoru [4].

3.3. Gesta

Kromě pohybu kurzorem po obrazovce je důležité uživateli umožnit složitější interakci s aplikací pomocí gest. Gesta by měla být pro uživatele intuitivní a jasně definovaná. Například pro potvrzení volby je přirozenější gesto podržení dlaně ovládající kurzor na místě, než gesto zvednutí druhé ruky nad úroveň hlavy.

3.3.1. Potvrzení volby

Nejčastější gesto pro uživatele je potvrzení volby poté, co posunul kurzorem.

Jako intuitivní gesto pro potvrzení volby se nabízí sevření dlaně. To odpovídá stavu, kdy uživatel chce vybranou věc uchopit do dlaně. Pro toto gesto je třeba rozlišit dva stavy dlaně – otevřená nebo zavřená. Senzor, a SDK k němu dodávané, tyto stavy dlaně nesleduje. Jediné, co je k dispozici, je orientační pozice dlaně v prostoru. Tato pozice přibližně odpovídá těžišti dlaně.

Snazší varianta pro potvrzení volby je podržet dlaň na stejném místě po zvolenou dobu. Důležitým prvkem pro toto gesto je zpětná vazba uživateli, aby po celou dobu věděl, že gesto provádí správně. Tato zpětná vazba může být realizována například kružnicí okolo kurzoru, která se postupně naplňuje, jak plyne potřebný čas pro dokončení gesta. Toto gesto není tolik náchylné na nepřesnosti, pokud je osoba ve větší vzdálenosti od senzoru.

3.3.2. Gesta posunu

Poté, co uživatel je schopen potvrdit výběr, je potřeba zajistit, aby mohl s výběrem pohybovat – například posunout vybranou molekulu ve zvoleném směru.

Pro gesto posunu je nutné určit sledovaný bod, směr, kterým se má posunout, a minimální vzdálenost, kterou musí vykonat.

3.3.3. Obecná gesta

Kromě specifických gest popsaných v předchozích kapitolách je vhodné rozpoznávat i složitější, obecná, gesta reprezentovaná posloupností více bodů těla v čase – například zamávání, výskok, apod.

Gesto rozpoznávající zamávání dlaní se dá využít pro pokračování v pozastavené hře.

3.4. Gesta pro implementaci

Gesta zvolená pro přirozené uživatelské rozhraní hry *Atomix* implementované v této práci jsou popsané v tabulce 3.

Akce	**Zvolené gesto**
Pohyb kurzoru	Relativně mapovaný při respektování PHIZ.
Potvrzení volby	Podržení dlaně na stejném místě po zvolený časový interval.
Pohyb atomem	Gesto posunu v daném směru dlaní, která ovládá kurzor.
Zamávání dlaní	Pokračování v pozastavené hře.

Tabulka 3: Zvolená gesta pro ovládání.

4. Rozpoznávání obecných gest

Za obecné gesto lze považovat libovolnou posloupnost zaznamenaných pozic předem zvolených sledovaných bodů za určitý časový úsek. U takto zaznamenaných gest je nutné zohlednit různou vzdálenost postav od senzoru, různou výšku postav a různou dobu trvání provedení gesta. Různou vzdálenost od senzoru a výšku osob lze sjednotit pomocí normalizace souřadnic na zvolenou jednotku. Jako jednotku lze zvolit například délku krku, tj. vzdálenost mezi středem trupu a hlavou.

Senzor *Kinect* zaznamenává data s jednotnou rychlostí (v ideálním případě 30 snímků za vteřinu), ale stejné gesto může různý člověk vykonat jinou rychlostí. Z tohoto důvodu pro porovnání podobnosti mezi posloupnostmi vzorového a zaznamenáno gesta není vhodné použít euklidovskou vzdálenost, která pro porovnání vyžaduje, aby obě zaznamenané posloupnosti byly stejně dlouhé [10].

Tento nedostatek lze vyřešit použitím algoritmu *Dynamic Time Warping* (DTW), který umožňuje porovnat podobnost dvou sekvencí o různé délce. Algoritmus DTW byl poprvé uveřejněn v 60. letech [11] a následně často využíván při rozpoznávání hlasu [12], ale tento princip nachází využití i v dalších oblastech [11]. Varianta algoritmu DTW je použita například ve hře *Kinect Dance Central* od společnosti *Microsoft* pro rozpoznávání tanečních pohybů [13].

Pro rozpoznávání gest senzorem *Kinect* existuje veřejně dostupná implementace algoritmu DTW[10], která však podporuje pouze gesta pevné délky, vždy používá všechny senzorem sledované body postavy a není kompatibilní s finální verzi Kinect SDK. Z těchto důvodů bylo nezbytné algoritmus nastudovat a implementovat jeho obecnější variantu.

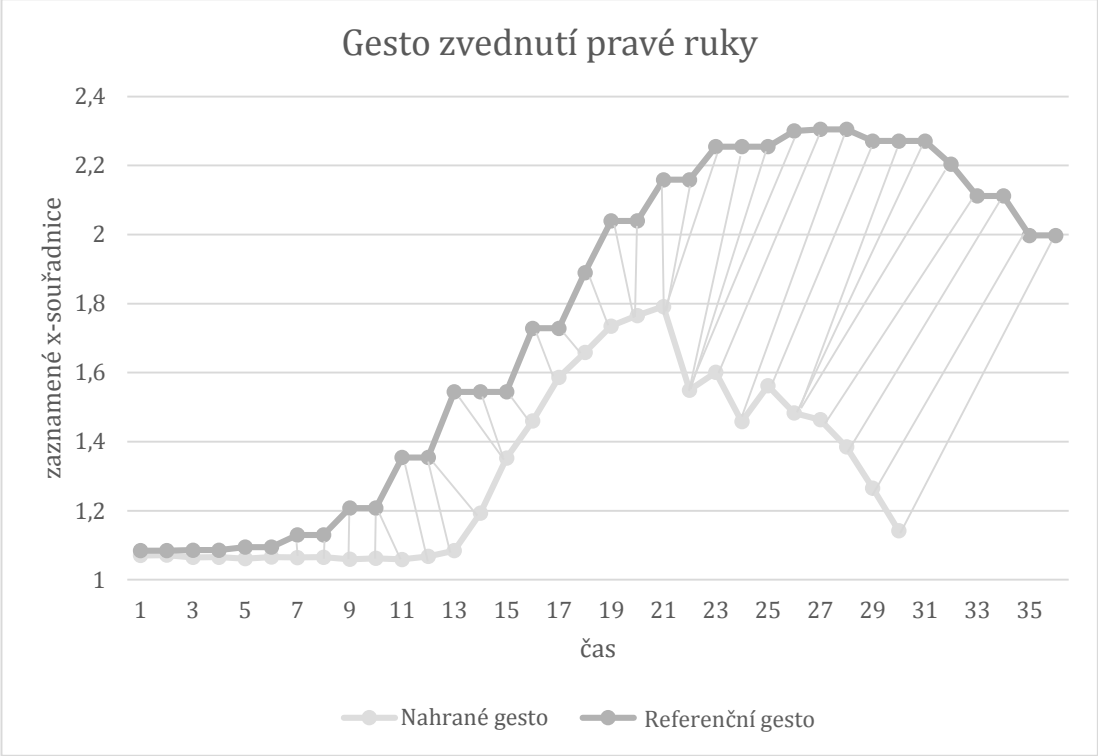
4.1. Algoritmus Dynamic Time Warping

Cílem algoritmu DTW je porovnat dvě různě dlouhé časově závislé posloupnosti $X = (x_1, x_2, \dots, x_m), m \in \mathbb{N}$ a $Y = (y_1, y_2, \dots, y_n), n \in \mathbb{N}$ pomocí nelineárního mapování – *warping*.

Nechť $\mathcal{F}$ obsahuje prvky obou posloupností: $\{x_i \mid 1 \leq i \leq m\} \subseteq \mathcal{F}, \{x_i \mid 1 \leq i \leq m\} \subseteq \mathcal{F}$. Dva prvky $\mathcal{F}$ jsou porovnány pomocí funkce lokální vzdálenosti $c : \mathcal{F} \times \mathcal{F} \to \mathbb{R}^+$. Intuitivně se funkce c volí taková, že hodnota $c(x, y)$ je malá, pokud jsou prvky x, y podobné, a v opačném případě je hodnota $c(x, y)$ vysoká.

[10] https://kinectdtw.codeplex.com/ (navštíveno 13. 7. 2014)

Příklad nelineárního mapování bodů dvou různě dlouhých posloupností při jejich porovnávání algoritmem DTW je na obrázku 9.



Obrázek 9: Párování DTW pro zaznamenané gesto.

Spočítáním lokální vzdálenosti mezi všemi prvky posloupností X a Y se získá matice vzdáleností $D = \{d_{i,j}\}_{1 \le i \le m, 1 \le j \le n} \in \mathbb{R}^{m \times n}$, kde $d_{i,j} \coloneqq c(x_i, y_j)$.

V této matici se hledá mapovací cesta – *warping path* –, tj. posloupnost $w = w_1, w_2, \dots, w_l, \dots, w_k$, kde l-tý prvek w je $w_l \coloneqq (u_i, v_j)_l$, kde $i \in \{1, \dots, m\}, j \in \{1, \dots, n\}, l \in \{1, \dots, k\}$, která splňuje následující podmínky:

(1) **Úplnost**: $w_1 = (1,1)$ a $w_k = (m,n)$,
(2) **Monotónnost**: $u_1 \le u_2 \le \cdots \le u_k$ a $v_1 \le v_2 \le \cdots \le v_k$,
(3) **Velikost kroku**: $w_{l+1} - w_l \in \{(1,0), (0,1), (1,1)\}$ pro $l \in \{1, \dots, k-1\}$.

Tato cesta určuje mapování sekvencí X a Y mezi sebou, podmínka úplnosti (1) vyžaduje, aby první a poslední prvky obou posloupností byly mapovány na sebe, tedy mapování zahrnuje celé sekvence. Podmínka monotónnosti (2) zajišťuje časovou souslednost mapování, pokud prvek v X předchází druhý, tak totéž platí pro odpovídající prvky v posloupnosti Y. Podmínka na velikost kroku (3) zajišťuje spojitost mapování, tedy že žádný prvek není během mapování vynechán.

Celková cena $c_w(X, Y)$ cesty w mezi posloupnostmi X a Y je za využití funkce lokální vzdálenosti definována jako:

$$c_w(X, Y) \coloneqq \sum_{r=1}^{k} c(x_{u_r}, y_{v_r})$$

Optimální cesta w^* mezi X a Y je taková cesta, která má minimální možnou cenu. DTW vzdálenost mezi posloupnostmi X a Y je definována jako cena optimální cesty mezi posloupnostmi X a Y:

$$DTW(X, Y) \coloneqq c_{w^*}(X, Y) = \min\{c_p(X, Y) \mid p \text{ je mapovací cesta mezi } X \text{ a } Y\}$$

K nalezení optimální cesty lze otestovat všechny cesty splňujících podmínky (1), (2) a (3), těchto cest ovšem existuje exponenciálně mnoho k délce vstupních posloupností, což vede k exponenciální časové složitosti výpočtu.

Příklad optimální cesty v matici vzdáleností splňující všechny kladené podmínky je na obrázku 10.

Obrázek 10: Grafická ilustrace průběhu algoritmu DTW.Matice lokálních vzdáleností s optimální cestou, vlevo průběh zaznamenané posloupnosti, nahoře průběh referenční posloupnosti.

Existují různé varianty algoritmu DTW pro zefektivnění výpočtu přidávající další omezující podmínky kladené na optimální cestu [12].

4.2. Výpočet pomocí dynamického programování

K výpočtu algoritmu DTW existuje algoritmus s časovou a prostorovou složitostí $O(M \cdot N)$ [14], který je založen na principu dynamického programování. Při tomto postupu se vzdálenost postupně akumuluje, pro každou položku matice vzdáleností se sčítá vzdálenost prvků na dané pozici s nejmenším předcházejícím prvkem.

Pro postupné sestavení matice vzdáleností $D \in \mathbb{R}^{m+1 \times n+1}$ je použit tento předpis [11]:

1. řádek: $d(1,i) = \infty, i \in \{1, \dots, m+1\}$,
1. sloupec: $d(i,1) = \infty, i \in \{1, \dots, n+1\}$,
1. prvek: $d(1,1) = 0$,
Ostatní prvky: $d(i,j) \coloneqq c(x_i, y_j) + \min\{d(i-1,j-1), d(i-1,j), d(i,j-1)\}$.

Poté v posledním prvku matice $d(m+1, n+1)$ je výsledná celková cena optimální cesty mezi posloupnostmi X a Y. Průběh optimální cesty mezi těmito posloupnostmi lze získat zpětným průchodem v takto vypočítané matici.

Pseudokód tohoto algoritmu je v uveden v ukázce 4.

```
DTW(X: [1..m], Y: [1..n]):
 dtw[] := new [n × m]
 dtw[0, 0] := 0

 for i := 1; i ≤ n; i++ do
    dtw[1, i] := ∞
 end for

 for j := 1; j ≤ m; j++ do
    dtw[j, 1] := ∞
 end for

 for i := 1; i ≤ n; i++ do
    for j := 1; j ≤ m; j++ do
       dtw[i, j] := c(i, j) +
                    min { dtw[i – 1, j]; dtw[i, j - 1]; dtw[i – 1, j - 1] }
     end for
 end for
 return dtw[n, m]
```

Ukázka 4: Pseudokód algoritmu pro výpočet DTW za využití dynamického programování.

5. Implementace

Tato kapitola popisuje, jakým způsobem je hra *Atomix* a související knihovny implementovány. Pro implementaci byla použita platforma Microsoft .NET Framework za použití jazyka C#. Pro vývoj hry byl využit Microsoft XNA Framework ve verzi 4.0 Refresh[11] respektive projekt MonoGame[12].

Hra je programována pro platformu Windows. Pomocné aplikace jsou programovány pro platformu Windows s použitím grafické knihovny Windows Presentation Foundation (WPF). Pro tyto aplikace byl využit návrhový vzor Model-View-ViewModel (MVVM).

5.1. XNA Framework

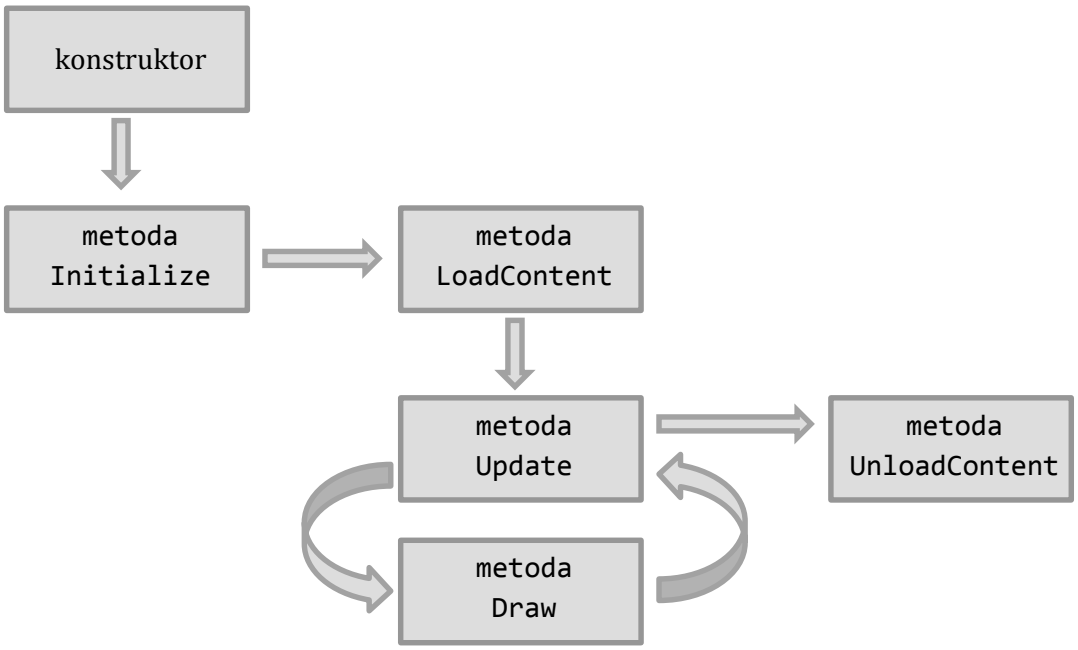
XNA Framework je platforma pro vývoj her pro Windows, XBOX 360 a Windows Phone 7. Framework už není společností *Microsoft* aktuálně vyvíjen[13], ale je nadále podporován.

Poslední vydanou verzí je XNA Framework 4.0 Refresh, která podporuje vývojové prostředí Microsoft Visual Studio pouze do verze 2010. Existují ovšem alternativní možnosti, jak zpřístupnit podporu pro XNA projekty i v novějších verzích programu Visual Studio. Komunita vytvořila rozšíření[14], které tuto podporu přidává i do aktuální verze 2013.

Tato knihovna usnadňuje vývoj her tím, že poskytuje správu životního cyklu hry a model pro herní komponenty. Stačí tak implementovat samotnou logiku hry pro jednotlivé fáze životního cyklu [15].

Životní cyklus platformy XNA Framework je znázorněn na obrázku 11. Všechny metody životního cyklu jsou virtuální a lze je ve vlastní implementaci přepsat.

[11] http://www.microsoft.com/en-us/download/details.aspx?id=23714 (navštíveno 20. 7. 2014)
[12] http://www.monogame.net/ (navštíveno 20. 7. 2014)
[13] http://www.pivotpointresearch.com/microsoft-exec-addresses-strategy-topic-frustrating-nearly-40-software-developers/ (navštíveno 26. 7. 2014)
[14] https://msxna.codeplex.com/ (navštíveno 20. 7. 2014)



Obrázek 11: Životní cyklus v XNA.

Metoda `Initialize` slouží k připravení hry, tj. nastavení hodnot proměnných, vytvoření instancí objektů apod.

Metoda `LoadContent` je volána po metodě `Initialize` poté, co je připraveno grafické zařízení. Případně kdykoliv během hry, pokud se změní grafické nastavení a je potřeba aktualizovat načtené textury, například při změně rozlišení hry. V této metodě by se měly načítat všechny grafické podklady potřebné pro vykreslení hry na obrazovku.

Po skončení inicializace hra pokračuje přechodem do herní smyčky, kterou zajišťují metody `Update` a `Draw`. Metoda `Update` by měla zpracovat veškerou logiku potřebnou k vykreslení tak, aby metoda `Draw` pouze překreslila obrazovku.

Po ukončení herní smyčky je nakonec zavolána metoda `UnloadContent`, která slouží k uvolnění načtených grafických podkladů.

Tento životní cyklus je shodný jak pro hru samotnou, tak pro jednotlivé herní komponenty. Hlavní třída hry dědí od třídy `Game`, herní komponenta dědí od třídy `GameComponent`, pokud obsahuje pouze logiku resp. od `DrawableGameComponent`, pokud se komponenta má i vykreslovat na obrazovku.

U jednotlivých komponent lze případně úplně vynechat předpřipravenou implementaci, kterou poskytuje XNA Framework, a implementovat rozhraní `IGameComponent`, `IUpdateable` pro komponentu pouze s logikou, případně navíc i rozhraní `IDrawable` pokud se komponenta má i vykreslovat [15].

5.2. MonoGame

Snahou projektu MonoGame je implementovat platformu pro vývoj her nad rozhraním DirectX, která má rozhraní pro programování (API) co nejvíce kompatibilní s platformou XNA Framework. Projekt MonoGame má otevřený zdrojový kód a je vyvíjen komunitou nezávisle na společnosti Microsoft. Oproti původní platformě XNA Framework MonoGame podporuje více operačních systémů a podporuje například i vývoj pro Android.

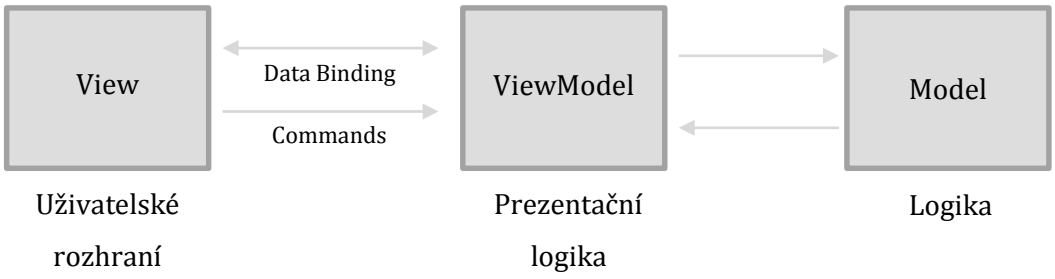
Pro knihovny implementované v rámci této diplomové práce pro platformu XNA Framework jsou taktéž vytvořeny i odpovídající projektové soubory využívající MonoGame, které používají identické zdrojové soubory.

5.3. Návrhový vzor MVVM

Návrhový vzor Model-View-ViewModel (MVVM) podobně jako vzor Model-View-Controller (MVC) pomáhá oddělit logiku aplikace od uživatelského rozhraní.

Poprvé tento návrhový vzor zveřejnil v příspěvku na svém blogu John Gossman[15] v roce 2005. MVVM vychází z obecnějšího návrhového vzoru Presentation Model (PM), který navrhl v roce 2004 Martin Fowler[16], a upravuje jej pro specifické potřeby platformy WPF [16].

Způsob oddělení a komunikace mezi vrstvami je znázorněn na obrázku 12.



Obrázek 12: Návrhový vzor MVVM.

Logika aplikace je kompletně oddělena od uživatelského rozhraní pomocí mezivrstvy s prezentační logikou. Jednotlivé proměnné jsou z prezentační logiky načítány pomocí *Data Binding* a složitější akce vykonávány přes příkazy (*Commands*). Díky tomu se nemusí prezentační logika starat o aktualizování uživatelského rozhraní [16].

[15] http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx (navštíveno 22. 7. 2014)
[16] http://martinfowler.com/eaaDev/PresentationModel.html (navštíveno 22. 7. 2014)

Prezentační logika nevyžaduje žádnou referenci na uživatelské rozhraní. Podrobněji o tomto návrhovém vzoru pojednává publikace [17].

5.4. Rozdělení na projekty

Zdrojový kód je rozdělen na několik samostatných projektů pro snadnější orientaci a především znovu použitelnost. Některé komponenty, které vznikly během programování hry, jsou zobecněné, aby nebyly přímo závislé na vyvíjené hře, a jsou vyčleněny do samostatných knihoven pro použití v dalších aplikacích.

Implementace se skládá z následujících projektů:

Mach.Wpf.Mvvm

Knihovna obsahující pomocné třídy pro implementaci návrhového vzoru MVVM, podrobněji v kapitole 6.1.

Mach.Kinect

Knihovna pro správu připojeného senzoru *Kinect* a s podporou detekce gest, podrobněji v kapitole 6.2.

Mach.Xna

Knihovna pro XNA Framework, která obsahuje grafické komponenty, správu obrazovek a jednoduchou správu vstupu od uživatele, podrobněji v kapitole 6.3.

Mach.Xna.Kinect

Knihovna pro XNA Framework, která rozšiřuje knihovnu *Mach.Xna* o komponenty využívající senzor *Kinect*, více v kapitole 6.4.

Mach.Xna.Content

Projekt obsahující grafický obsah, který je používán v XNA knihovnách.

Mach.Xna.Content.Compiler

Prázdný projekt knihovny, jehož jediným účelem je, aby zkompiloval grafický obsah umístěný v projektu *Mach.Xna.Content*. Zkompilovaná grafika je následně připojena jako součást knihoven *Mach.Xna* a *Mach.Xna.Kinect* pro jejich snadnou distribuci.

Mach.Kinectomix.Logic

Knihovna s logikou pro hru *Atomix*. Knihovna je využita jak ve hře, tak v editoru úrovní, podrobněji v kapitole 7.1.

Mach.Kinectomix

Projekt s implementací hry *Atomix* za využití herního XNA Frameworku, podrobněji v kapitole 7.2.

Mach.Kinectomix.Content

Projekt obsahuje grafický obsah, který využívá implementace hry v projektu *Mach.Kinectomix*.

Mach.Kinectomix.Editor

Aplikace pro editaci úrovní pro hru *Atomix*, podrobněji v kapitole 7.3.

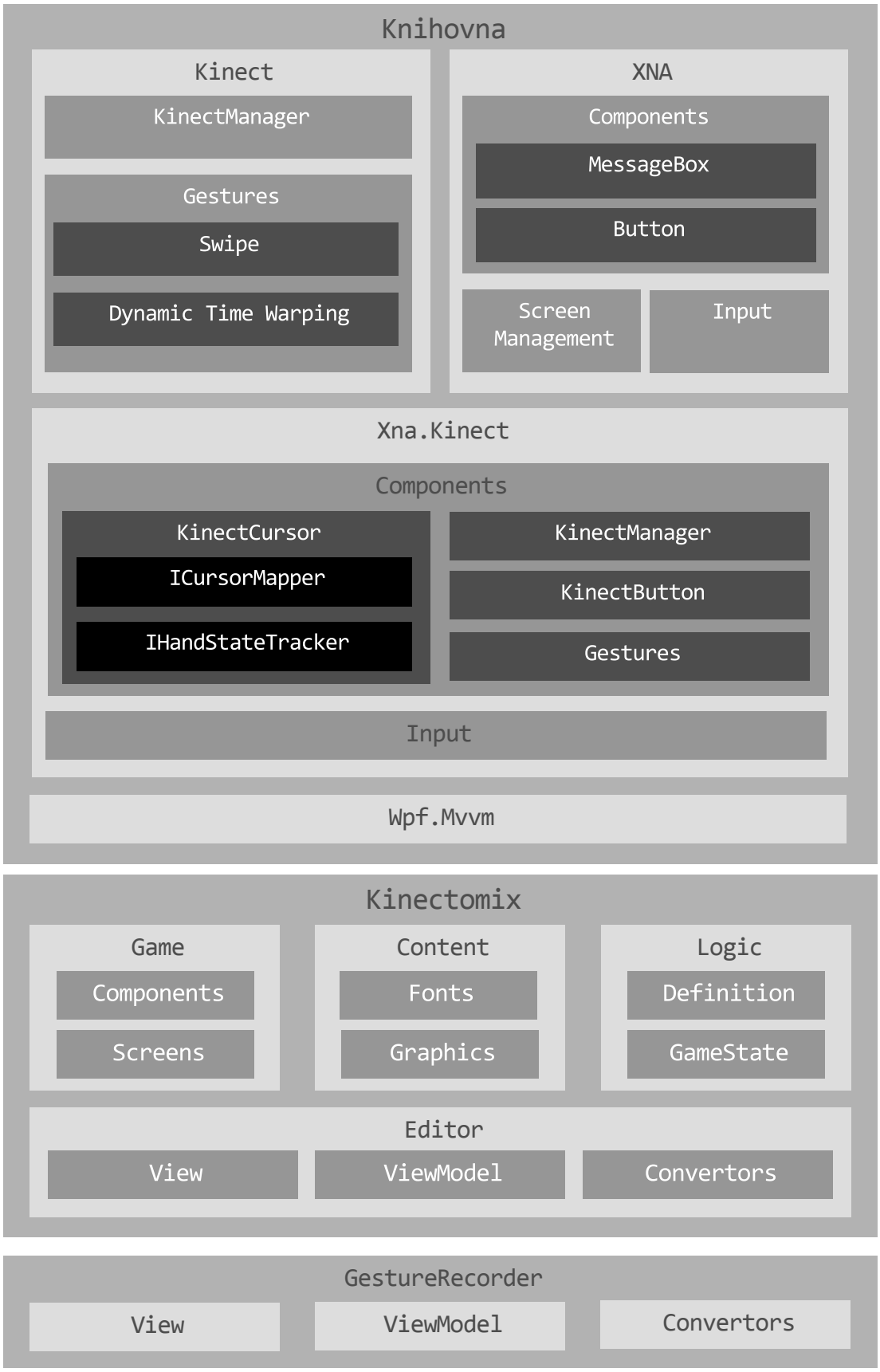
Mach.Kinect.GestureRecorder

Aplikace umožňující zaznamenání průběhu gesta pro jeho následné rozpoznávání, podrobněji v kapitole 7.4.

Celkovou architekturu implementace ilustruje diagram na obrázku 13, zanoření položek v diagramu odpovídá jmenným prostorům případně třídám.

Všechny implementované projekty jsou lokalizovány do českého a anglického jazyka.

Implementaci knihoven popisuje kapitola 6, implementaci samotné hry a pomocných aplikací se věnuje následující kapitola 7.



Obrázek 13: Rozdělení dle modulů.

6. Implementace knihoven

Tato kapitola popisuje implementaci knihoven *Mach.Wpf.Mvvm*, *Mach.Kinect*, *Mach.Xna* a *Mach.Xna.Kinect*.

6.1. Knihovna MVVM

Knihovna *Mach.Wpf.Mvvm* obsahuje pomocné třídy pro knihovnu WPF usnadňující práci s návrhovým vzorem Model-View-ViewModel popsaným v kapitole 5.3.

Objektový model této knihovny je na obrázku 14.

Obrázek 14: Objektový model knihovny.

NotifyPropertyBase

Třída `NotifyPropertyBase` implementuje WPF rozhraní `INotifyPropertyChanged`, které slouží k propagování změn ze zdrojových vlastností prezentační logiky do grafického rozhraní.

DelegateCommand

Třída `DelegateCommand` implementuje WPF rozhraní `ICommand`, které slouží k volání logiky z grafického rozhraní. V rámci této implementace lze zaregistrovat akci, která se má vykonat, a funkci, která ověřuje, jestli je dovoleno akci provést. Implementace podporuje základní `DelegateCommand` i generickou variantu `DelegateCommand<T>`, která kontroluje typ předávaného parametru funkcím, které se mají vykonat.

IFileDialogService

Rozhraní `IFileDialogService` vytváří obecný model pro práci s dialogovými okny `FileOpenDialog` a `FileSaveDialog` při použití návrhového vzoru MVVM.

6.2. Knihovna Kinect

Knihovna *Mach.Kinect* obsahuje pomocné třídy pro práci se senzorem *Kinect*.

6.2.1. Správa připojeného senzoru

Kinect SDK podporuje připojení více senzorů současně. Pro usnadnění práce s připojováním senzorů je vytvořena třída `KinectManager`, která se stará o životní cyklus všech aktuálně připojených senzorů *Kinect*. Pokud detekuje nově připojený senzor, tak jej automaticky spustí, a podle nastavení třídy `KinectManager` případně spustí i odpovídající proudy dat – *Skeleton stream*, *Color stream*, *Depth stream*. Při vytváření instance třídy `KinectManager` je možné nastavit limit na počet připojených senzorů. Jakmile počet připojených senzorů dosáhne tohoto limitu, přestane `KinectManager` sledovat stav dalších. Tímto způsobem se dá zařídit, aby aplikace použila pouze jeden senzor, a další byl dostupný jiným aplikacím.

Pro úspěšně připojený senzor je vytvořena instance třídy `ConnectedSensor`, která obsahuje připojený `KinectSensor`. Kromě senzoru tato třída také zajišťuje pomocí třídy `Skeletons` i zpracování *Skeleton stream* ze senzoru a ze získaného `SkeletonFrame` následné vyhledávání sledované osoby podle zadané strategie. Implementované strategie pro sledování postav před senzorem jsou následující (s odpovídající hodnotou výčtového typu `SkeletonTrackingType`):

První `First`

První senzorem rozpoznaná postava, u které je známá alespoň pozice v prostoru, tedy postavy se stavem jak `PositionOnly`, tak i `Tracked`. Podrobněji viz kapitola 3.1.1 (První sledovaná osoba).

První aktivně sledovaná `FirstFullyTracked`

První senzorem rozpoznaná postava, která je senzorem aktivně sledována, tedy jen postavy se stavem `Tracked`. Tato strategie je brána jako výchozí.

Nejbližší `Nearest`

Postava, která je k senzoru nejblíže a je známá alespoň její pozice v prostoru. Podrobněji viz kapitola 3.1.3 (Nejbližší sledovaná osoba).

Nejbližší aktivně sledovaná `NearestFullyTracked`

Postava, která je k senzoru nejblíže a je senzorem aktivně sledována.

Podle zadaného ID `FixById`

Sledování pevně zvolené postavy podle jejího unikátního identifikátoru. Tento identifikátor je třeba nastavit pomocí vlastnosti `TrackedSkeletonId`. Podrobněji viz kapitola 3.1.2.

Podle zadaného ID nebo první `FixByIdOrFirst`

Sledování pevně zvolené postavy podle jejího unikátního identifikátoru, a pokud žádná taková postavy není nalezena, použije se první senzorem nalezená, u které je známá alespoň její pozice v prostoru.

Podle zadaného ID nebo první plně sledovaná `FixByIdOrFirstFullyTracked`

Sledování pevně zvolené osoby podle jejího unikátního identifikátoru, a pokud žádná taková osoba není nalezena, použije se první senzorem nalezená, která je senzorem aktivně sledována.

Nejbližší vždy aktivně sledovaná `NearestForcedFullyTracked`

Postava, která je k senzoru nejblíže a pokud je sledována pouze pasivně, změní režim sledování této postavy na aktivní.

Pro všechny strategie platí, že pokud se nenajde žádná vyhovující postava, tak je vrácena hodnota `null`.

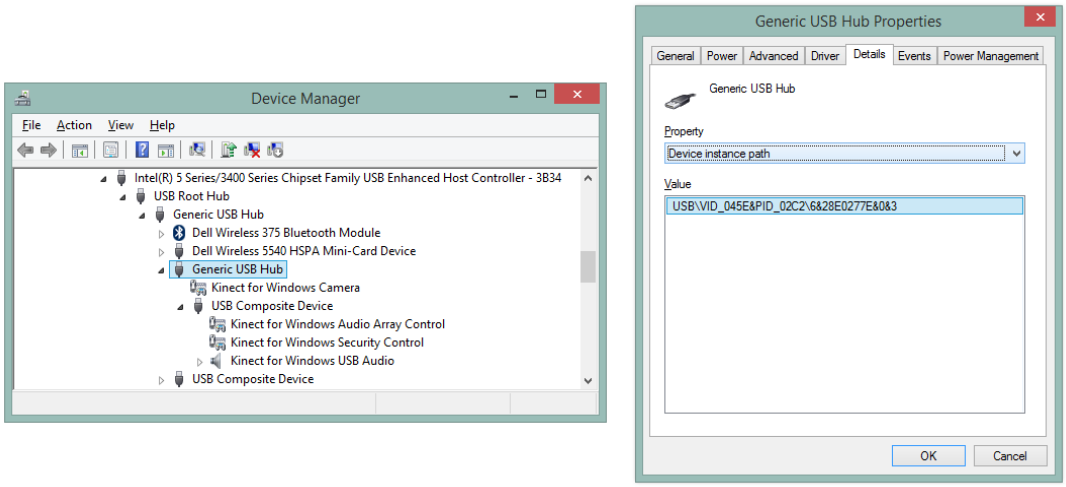
Zpracování sledovaných postav není automaticky zapnuto, pro automatické zpracování dat z *Skeleton stream* pomocí událostí je nutno nastavit hodnotu vlastnosti `ProcessSkeletonsAutomatically` na `true`. Tato vlastnost je jak na hlavní třídě `KinectManager` spravující všechny připojené senzory, tak na jednotlivých připojených senzorech reprezentovaných instancemi třídy `ConnectedSensor`.

Pro ruční zpracování, pokud se nevyužívá zpracování pomocí událostí, je nutno volat metodu `ProcessUpdate` na třídě `KinectManager`, která zajistí zpracování dat z *Skeleton stream* na všech připojených senzorech *Kinect* pomocí této třídy.

Výchozí chování senzoru je takové, že první dvě rozpoznané postavy sleduje aktivně, zbylé pasivně. Pokud tato strategie nevyhovuje, je možné senzoru explicitně přikázat, které až dvě postavy, podle unikátního identifikátoru, má sledovat aktivně pomocí metody `ChooseSkeletons` na třídě `SkeletonStream` [3]. Tento postup je použit u strategie *Nejbližší vždy aktivně sledovaná*.

Pro přístup k připojeným senzorům lze využít vlastností třídy `KinectManager` a vlastního indexeru, který vyhledá připojený senzor na základě identifikátoru

`DeviceConnectionId`, který unikátně identifikuje připojený senzor k počítači. Tento identifikátor je závislý na způsobu připojení senzoru, není to unikátní identifikátor senzoru jako takového. Pokud se senzor připojí do jiného USB portu na počítači, přidělený identifikátor se může změnit[17]. Tento identifikátor je možné zjistit ve Správci zařízení, v Zobrazení změnit na *Zařízení podle připojení*, a najít *Obecný rozbočovač USB* (případně *Generic USB Device*), které obsahuje zařízení s názvem začínající „*Kinect for Windows*“. Ve vlastnostech tohoto zařízení na záložce *Podrobnosti* je hodnota vlastnosti *Cesta instance zařízení* tento unikátní identifikátor pro připojený senzor. Na obrázku 15 je zobrazen snímek obrazovky Správce zařízení se zobrazeným identifikátorem senzoru.



Obrázek 15: Zjištění hodnoty DeviceConnectionId ve Správci zařízení.

6.2.2. Rozpoznávání gest

Implementace knihovny nabízí jednoduché rozpoznání gest posunu ve čtyřech základních směrech – doleva, doprava, nahoru, dolů – pomocí třídy `SwipeRecognizer` a rozpoznávání obecných gest pomocí třídy `Recorder`.

Gesta posunu

Rozpoznání gest posunu ve směru pomocí třídy `SwipeRecognizer` je implementováno tak, že pro každé rozpoznání gesta je nutné zavolat metodu `Start`, kde první parametr je startovací bod se souřadnicemi, od kterého se začne gesto rozpoznávat, a druhý parametr obsahuje minimální požadovanou vzdálenost, kterou musí bod vykonat, aby se gesto považovalo za splněné. Tato vzdálenost je očekávána v metrech. Po spuštění rozpoznávání je při každé aktualizaci polohy pomocí metody `ProcessPosition` kontrolován posun bodu a pokud je gesto rozpoznáno, je vráceno v druhém, výstupním,

[17] http://msdn.microsoft.com/library/microsoft.kinect.kinectsensor.deviceconnectionid.aspx (navštíveno 18. 7. 2014)

parametru. Pokud rozpoznávání selhalo, návratová hodnota této funkce je `false` a rozpoznávání je ukončeno.

Při přidání nové pozice se kontroluje, jestli se bod posouvá v předpokládaném směru. Pro horizontální variantu gesta (posun doleva nebo doprava) se bod musí pohybovat ve směru osy x, tedy se sčítá posun v ose x, a kontroluje se, jestli bod neopustil nastavenou toleranci v posunu na osách y, resp. z (z vlastností `VerticalTolerance`, resp. `DepthTolerance`). Pro vertikální variantu gesta (posun nahoru nebo dolů) se bod musí pohybovat ve směru osy y, tedy se sčítá posun v ose y a kontroluje se, jestli bod neopustil nastavenou toleranci na osách x, resp. z (z vlastností `HorizontalTolerance`, resp. `DepthTolerance`).

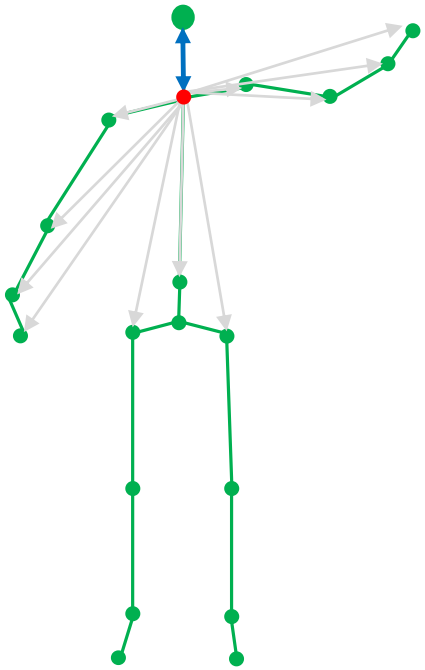
Třída `SwipeGesture` s rozpoznaným gestem obsahuje směr, ve kterém bylo gesto rozpoznáno (vlastnost `Direction`) a vzdálenost (vlastnost `Distance`) v metrech, kterou sledovaný bod vykonal, než dosáhl požadovaného minima nastaveného metodou `Start`.

Obecná gesta

Pro implementaci rozpoznávání obecných gest byl využit algoritmus DTW, který je popsán v kapitole 4.1. Za normalizační jednotku vzdálenosti bodů postavy byla zvolena délka krku, tj. vzdálenost mezi hlavou – bod `Head` a středem trupu – bod `ShoulderCenter`.

Pro potřeby výpočtu DTW je jeden prvek sekvence gesta reprezentován třídou `GestureFrame`. Ta obsahuje vektory vzdáleností jednotlivých bodů těla od bodu `ShoulderCenter` normalizované délkou krku v pořadí daném výčtovým typem `JointType`. Grafické znázornění těchto vektorů je na obrázku 16. Sekvence prvků je reprezentována třídou `Gesture` obsahující seznam instancí `GestureFrame`.

Takto zvolená reprezentace bodů následně umožňuje snadný výpočet vzdálenosti mezi dvěma zaznamenanými rámci pomocí funkce lokální vzdálenosti *c* implementovanou funkcí `FrameDistance` ve statické třídě `DynamicTimeWarping`. Funkce vrací ohodnocení vzdálenosti dvou `GestureFrame` jako součet euklidovských vzdáleností všech sledovaných bodů postavy vzájemně mezi dvěma `GestureFrame`. Seznam těchto sledovaných bodů je uložen ve vlastnosti `TrackedJoints` v instanci třídy `Gesture`. Předpokládá se, že jednotlivé pozice sledovaných bodů postavy v obou rámcích jsou ve stejném pořadí, které určuje výčtový typ `JointType`. Výpočet ceny optimální cesty mezi dvěma gesty zajišťuje statická funkce `CalculateDistance` ve třídě `DynamicTimeWarping`.



Obrázek 16: Zpracování postavy pro DTW se sledovanými body horní části těla. Středový bod je označen červeně, vektory vzdáleností mezi body a středem šedě, normalizační jednotka modře.

Samotné rozpoznávání gest implementuje třída `Recognizer`. Pro zvýšení efektivity se nejprve porovnává aktuálně zaznamenaný `GestureFrame` s posledními rámci vzorových gest a jen pokud jsou si tyto rámce podobné, tak se u nich spustí kompletní výpočet DTW nad celým zaznamenaným průběhem gest. V jednu chvíli může být rozpoznáno více gest současně.

Normalizaci souřadnic zaznamenaných bodů zajišťuje nadřazená abstraktní třída `GestureProcessor`. Od této třídy dědí i třída `Recorder`, která zajišťuje záznam nových gest. Pro každé gesto lze nastavit, jestli se sleduje i pozice v prostoru nebo pouze v jedné rovině. U každého rozpoznaného gesta reprezentovaného instancí třídy `RecognizedGesture` je vrácena vypočtená DTW vzdálenost a jeho cena, což je DTW vzdálenost normalizovaná počtem zaznamenaných rámců kandidátního gesta.

6.3. Knihovna XNA

Knihovna *Mach.Xna* obsahuje následující komponenty použitelné při vývoji hry s pomocí platformy XNA Framework 4.0:

`Button`	textové tlačítko
`SpriteButton`	tlačítko pro textury
`FrameRateInfo`	výpočet aktuálního FPS hry

`MessageBox`	dialogové okno
`ScreenManager`	správce herních obrazovek

Tuto knihovnu lze zkompilovat i pro použití v XNA pro XBOX 360.

6.3.1. Vstup od uživatele

Součástí implementace je návrh modelu pro jednotné zpracování vstupu od uživatele z různých vstupních zařízení. Instance `IInputProvider` spravuje dané vstupní zařízení, jehož aktuální stav lze získat zavoláním metody `GetState`. Zjištěný stav je vrácen ve struktuře `IInputState`, která obsahuje souřadnice, kde se zařízení nachází, a aktuální hodnotu sledovaného stavu.

Tento model pro zpracování vstupu je použit při implementaci komponenty pro tlačítko, u které pro fungování stačí informace o aktuálních souřadnicích a jestli je nebo není sledovaný stav aktivní, například v případě myši, je-li zmáčknuté levé tlačítko.

Obrázek 17: Objektový model správy vstupu.

Knihovna implementuje zpracování vstupu ze senzoru *Kinect* a myši. Objektový model tříd zpracovávající vstup je na obrázku 17.

Dále je implementována třída `MutipleInputProvider`, která slouží ke společnému zpracování vstupů z více zařízení souběžně. Jednotlivá vstupní zařízení se registrují pomocí metody `AddProvider`. Pokud je zavolána metoda `GetState` pro získání aktuálního stavu sledovaných zařízení, jsou procházena jednotlivá registrovaná zařízení a vrácen stav prvního zařízení, které indikuje, že jeho sledovaný stav je aktivní. Například v případě zařízení myši jestli je stisknuté levé tlačítko. Pokud žádné zařízení neindikuje

aktivní stav, tak je vráceno první zařízení, které vrátí nenulovou pozici. Pokud ani takové zařízení se nenajde, je vrácen stav prvního registrovaného zařízení.

6.3.2. Tlačítko

Knihovna obsahuje implementace komponenty `Button` pro textové tlačítko a `SpriteButton` pro tlačítko využívající textury. Obě komponenty vychází z abstraktní třídy `ButtonBase` obsahující společnou logiku.

ButtonBase

Abstraktní třída `ButtonBase` podporuje zpracování vstupu od uživatele pomocí modelu popsaného v kapitole 6.3.1. Pokud je zjištěno aktivování tlačítka je vyvolána událost `Selected`. Pro uložení vlastních dat svázaných s tlačítkem je k dispozici vlastnost `Tag`.

Button

Komponenta `Button` podporuje vykreslení textu a graficky rozlišuje, jestli vstupní zařízení hlásí pozici nad ním nebo ne pomocí změny barvy pozadí. Jména vlastností ovlivňující vzhled tlačítka kopírují jmennou konvenci, kterou používá tlačítko v grafické knihovně WPF. Podporuje různá horizontální zarovnání textu nastavitelná pomocí vlastnosti `TextAlignment` a posouvání textu uvnitř tlačítka, pokud se do tlačítka nevejde celý. Pro posouvání textu jsou implementovány dvě strategie nastavitelné vlastností `TextScrolling`. První strategie posouvá textem zprava doleva stále dokola, druhá směry střídá.

Do rozměrů – `Width`, `Height` – se nepočítá šířka rámečku `BorderThickness` ani `Padding`, pro snazší zjištění výsledného rozměru tlačítka jsou k dispozici vlastnosti `ActualWidth` a `ActualHeight`.

SpriteButton

Komponenta `SpriteButton` podporuje vykreslení různých textur podle aktuálního stavu tlačítka.

6.3.3. Dialogové okno

Komponenta implementující dialogové okno pro XNA Framework funkčně podobné tomu, které je známé z klasických aplikací pro stolní počítače. Aplikační rozhraní (API) je pro usnadnění práce podobné rozhraní dialogového okna, které obsahuje grafická knihovna WPF.

Hlavní logika dialogového okna je implementována v abstraktní třídě `MessageBoxBase`. Z této abstraktní třídy vychází třída `MessageBox`, která používá pro tlačítka komponentu `Button`.

Pro zobrazení dialogu slouží metoda `Show`, která má tři přetížení. První varianta s jedním parametrem nastavuje text, který se má zobrazit s tlačítkem OK. Druhá varianta akceptuje text, který se má zobrazit, a typ sady tlačítek, které se mají zobrazit, jednotlivé sady jsou definované pomocí výčtového typu `MessageBoxButtons`. Třetí nejobecnější varianta akceptuje text, který se má zobrazit, a pole libovolných tlačítek, které se zobrazí. U všech variant se jako výsledek zobrazení dialogového okna bere hodnota z výčtového typu `MessageBoxResult`, která je u každého tlačítka uložena ve vlastnosti `Tag`. Na rozdíl od grafické knihovny WPF tato implementace výsledek nevrací jako hodnotu volání metody `Show`, ale pomocí vyvolání události `Changed`. Hodnota vybraného tlačítka je předána ve vlastnosti `Result` při obsluze události.

Objektový model pro práci s dialogovými okny je znázorněn na obrázku 18.

Obrázek 18: Objektový model pro práci s dialogovým oknem.

Příklad vytvoření dialogového okna s vlastní sadou tlačítek je v ukázce 5.

```
// Vytvoření vlastních tlačítek
Button continueButton = new Button(this)
{
    Tag = MessageBoxResult.Yes, Content = "Pokračovat k další úrovni",
};
Button exitButton = new Button(this)
{
    Tag = MessageBoxResult.No, Content = "Ukončit hru",
};

MessageBox messageBox = new MessageBox(this, inputProvider);

// Zaregistrování události vyvolané po stisknutí libovolného tlačítka
messageBox.Changed += messageBox_Changed;
// Zobrazení dialogu
messageBox.Show("Co si přejete udělat?",
    new Button[] { continueButton, exitButton });
```

Ukázka 5: Vyvolání dialogu s vlastní sadou tlačítek.

6.3.4. Správa herních obrazovek

XNA Framework neobsahuje žádné pomocné třídy pro správu více obrazovek hry. Knihovna proto implementuje model pro správu herních obrazovek. Model se skládá ze základní obrazovky implementované abstraktní třídou `GameScreen` a ze správce obrazovek implementovaným třídou `ScreenManager`.

Třída `ScreenManager` zajišťuje správu jednotlivých herních obrazovek, tj. vykreslení aktivní obrazovky a přepínání mezi nimi. Vychází z třídy `DrawableGameComponent`.

Abstraktní třída `GameScreen` implementuje základní funkcionalitu obrazovky, metody pro životní cyklus hry v XNA frameworku a správu herních komponent. Herní obrazovka nabízí stejný způsob práce s komponentami jako třída `Game` XNA frameworku pomocí kolekce `GameComponentCollection` dostupné přes vlastnost `Components`. Každá obrazovka smí být přiřazena pouze do jednoho správce obrazovek. Správce obrazovek, který se stará o danou obrazovku, je uvnitř obrazovky přístupný přes vlastnost `ScreenManager`. Obrazovka má přístup k nadřazenému objektu hry, která komponentu obsahuje, pomocí svého správce obrazovek. Ten obsahuje vlastnost `Game` odkazující na instanci třídy `Game`.

Pro vykreslení obrazovky je třeba ji nejprve aktivovat v přiřazeném správci metodou `Activate`. Po aktivaci je na třídě obrazovky zavolána metoda `Activated`. Při odebrání obrazovky z vykreslování je zavolána metoda `Deactivated`. Díky těmto metodám mohou obrazovky reagovat na změny ve vykreslování na obrazovku.

6.4. Knihovna Xna.Kinect

Knihovna *Mach.Xna.Kinect* rozšiřuje knihovnu *Mach.Xna* popsanou v kapitole 6.3. o následující komponenty, které přímo využívají senzoru *Kinect*:

`VisualKinectManager`	komponenta zobrazující stav připojeného senzoru
`KinectCursor`	kurzor ovládaný senzorem Kinect
`KinectCircleCursor`	kurzor ovládaný senzorem Kinect podporující zobrazení kružnice se stavem okolo kurzoru
`KinectButton`	textové tlačítko reagující na kurzor ovládaný senzorem
`KinectSpriteButton`	tlačítko pro textury reagující na kurzor ovládaný senzorem
`KinectMessageBox`	dialogové okno využívající tlačítka reagující na senzor
`Gestures`	komponenta rozpoznávající obecná gesta
`ClippedEdgesVisualiser`	komponenta zvýrazňující hrany obrazovky u kterých je senzorem sledovaná postava moc blízko
`SkeletonRenderer`	komponenta vykreslující senzorem rozpoznané postavy
`VideoStreamComponent`	komponenta vykreslující video výstup ze senzoru
`VideoWhenNoSkeleton`	komponenta vykreslující video výstup ze senzoru pokud není žádná sledovaná postava k dispozici

6.4.1. Kurzor

XNA komponenta `KinectCursor` zajišťuje výpočet a vykreslení vlastního kurzoru na obrazovce hry, kromě vykreslení vlastní textury kurzoru lze povolit upravování pozice systémového kurzoru nebo úplně zakázat vykreslování vlastní textury. Výchozí nastavení instance třídy `KinectCursor` vykresluje texturu odpovídající pozici dlaně a systémový kurzor je ponechán zobrazen na své původní pozici.

Kód v ukázce 6 nastavuje, že bude naopak používán pouze systémový kurzor.

```
KinectCircleCursor cursor = new KinectCircleCursor(this, kinectManager);
cursor.Texture = null;
cursor.HideSystemCursorWhenHandTracked = false;
cursor.UpdateSystemCursorPosition = true;
```

Ukázka 6: Senzorem ovládaný kurzor zobrazující pouze systémový kurzor.

U kurzoru lze nastavit, zda má sledovat pouze levou dlaň, pravou dlaň nebo automaticky rozhodovat mezi oběma. Automatické rozhodování použije tu dlaň, která je k senzoru blíže. Výchozí chování je automatická detekce. Způsob sledování se nastavuje pomocí vlastnosti `TrackedCursorHandMode`.

6.4.1.1. Pozice kurzoru

Výpočet aktuální pozice kurzoru je zobecněn pomocí rozhraní `ICursorMapper`. Tento způsob umožňuje snadno zaměnit implementaci mapování kurzoru bez nutnosti změny samotné komponenty. Mapování kurzoru se aktivuje pomocí vlastnosti `CursorMapper` na třídě `KinectCursor`. Aktuální pozici kurzoru na obrazovce zpřístupňuje vlastnost `Position` a aktuální pozici dlaně v prostoru, která je kurzorem sledována zpřístupňuje vlastnost `HandPosition`.

Knihovna obsahuje dvě implementace mapování. Objektový model je znázorněn na obrázku 19.

Obrázek 19: Objektový model mapování kurzoru.

Absolutní mapování

Základní mapování, kde se pozice dlaně v prostoru přepočítá ve stejném poměru na obrazovku:

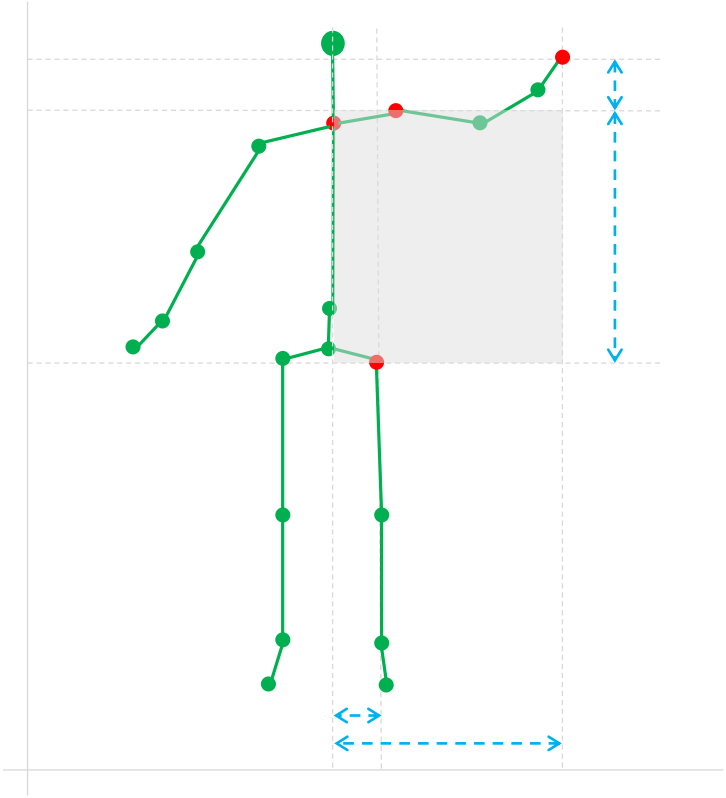
$$cursor_x = \frac{hand_x}{video_{width}} \cdot screen_{width}\,, \qquad cursor_y = \frac{hand_y}{video_{height}} \cdot screen_{height}$$

kde $cursor$ je výsledná pozice kurzoru na obrazovce, $hand$ je pozice dlaně získaná ze senzoru, $video$ je rozměr videa získaného ze senzoru, ve kterém je obsažena pozice dlaně a $screen$ je rozměr obrazovky, na kterou se kurzor mapuje. Nedostatkům tohoto mapování se věnuje kapitola 3.2.1.

Toto mapování implementuje třída `AbsoluteCursorMapper`.

Relativní mapování

Relativní mapování se snaží řešit nedostatky absolutního mapování omezením se na body v rámci optimální interakční zóny (viz kapitola 3.2.2) místo celého prostoru snímaného senzorem. Horní hrana obrazovky je mapována na ramena osoby, dolní hrana obrazovky na boky a levý horní roh obrazovky odpovídá krku. Toto mapování znázorňuje obrázek 20, body na těle použité při mapování jsou vyznačeny červeně.



Obrázek 20: Relativní poměr pro mapování kurzoru.

Navrženému mapování odpovídá tento přepočet souřadnic:

$$cursor_x = \frac{right\_hand_x - shoulder\_center_x}{(right\_shoulder_x - shoulder\_center_x) \times 2} \cdot screen_{width}$$

$$cursor_y = \frac{right\_hand_y - right\_shoulder_y}{hip_y - right\_shoulder_y} \cdot screen_{height}$$

kde $cursor$ je výsledná pozice kurzoru na obrazovce, $screen$ je rozměr obrazovky, na kterou se kurzor mapuje a $right\_hand$, $shoulder\_center$, $right\_shoulder$, hip jsou pozice body těla vrácené senzorem.

Implementace tohoto mapování je ve třídě `RelativeCursorMapper`. Toto mapování je u komponenty `KinectCursor` nastaveno jako výchozí.

6.4.1.2. Vyhlazování kurzoru

Pro zajištění plynulejšího pohybu kurzoru po obrazovce jsou zaznamenané pozice kurzoru ukládány do pomocného pole, a výsledná pozice kurzoru je následně spočtena jako průměr posledních zaznamenaných pozic. Délka vyhlazovacího pole je nastavitelná pomocí vlastnosti `CursorPositionsBufferLength`. Vyhlazování se vypne nastavením délky vyhlazovacího pole na hodnotu `1`, což je výchozí hodnota.

6.4.1.3. Detekce stavu

Kromě pozice kurzoru komponenta také podporuje rozlišení určeného stavu, například otevřená/zavřená dlaň. Toto sledování stavu je zobecněno pomocí rozhraní `IHandStateTracker`, diagram tohoto rozhraní je na obrázku 21.

Obrázek 21: Rozhraní IHandStateTracker.

Současná implementace `KinectCursor` podporuje sledování pouze jednoho stavu. Stav, který má být sledován, se registruje pomocí vlastnosti `HandStateTracker`.

Otevřená/zavřená dlaň

Senzor vrací pouze pozici celé dlaně, nikoliv její stav. Pro rozhodnutí o stavu dlaně je tedy nutné nejdříve dlaň detekovat a rozhodnout o stavu. K tomu se dá využít algoritmus pro hledání hran v obrazu, podrobněji se této metodě detekce věnuje vědecký článek [18].

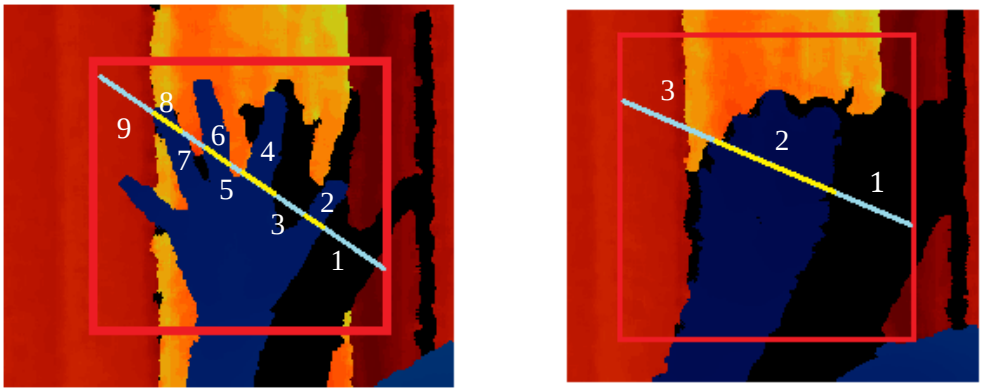
Jelikož ale senzor vrací pro každý pixel z *Depth stream* i identifikátor uživatele, a *Skeleton stream* pro bod dlaně používá stejný identifikátor, je možné oddělit postavu od zbytku snímaného obrazu zkombinováním těchto dvou datových proudů a není nutné provádět výpočetně náročnou detekci hran.

Ze získané celé postavy je následně oddělena jen plocha samotné dlaně. Pro detekci otevřené, resp. zavřené dlaně toto oddělení nemusí být nutně přesně v zápěstí a lze použít kruh se středem v bodě dlaně získaném ze senzoru. V tomto kruhu jsou vybrány pouze pixely obsahující unikátní identifikátor sledované postavy.

Jako poloměr kruhu je použita délka krku, jelikož délku krku lze snadno zjistit z pozic vrácených senzorem a průměr takto zvolené kružnice přibližně odpovídá délce otevřené dlaně. Pro zpřesnění se získaná plocha ještě omezuje nastavením tolerance na ose z, aby se do plochy dlaně nezohlednily části těla, které jsou za dlaní.

V implementaci je pro snazší výpočet místo kruhu použit čtverec. V takto získaném čtverci je možné provést samotnou detekci, jestli je dlaň zavřená nebo otevřená. Pokud je dlaň uzavřená v pěst, zaznamenaná plocha dlaně je konvexní, v opačném případě lze předpokládat, že dlaň je otevřená.

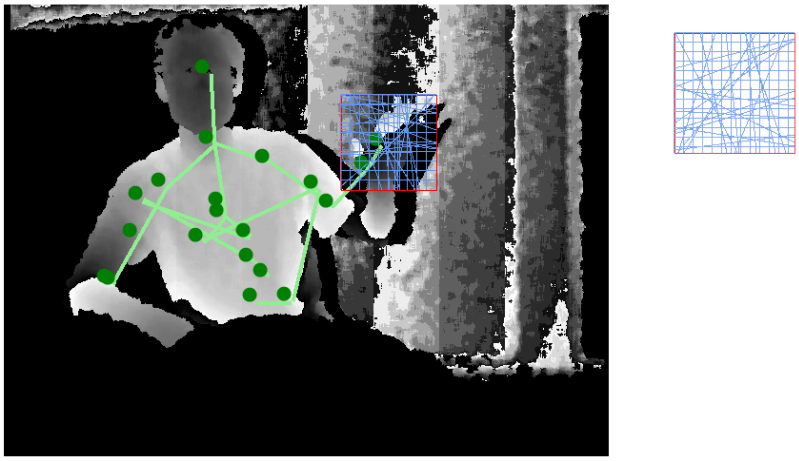
Detekce konvexnosti je provedena pomocí úseček náhodně rozmístěných po čtverci obsahujícím dlaň. Pokud existuje úsečka, která má více než 3 změny (tj. posloupnost změn na úsečce je jiná než volná plocha, dlaň, volná plocha), tak lze předpokládat, že plocha dlaně je konkávní, a tedy otevřená. Na obrázku 22 (a) je detekční úsečka, která obsahuje více než 3 změny (přechody jsou označeny žlutou barvou), na obrázku 22 (b) je úsečka, která má právě 3 změny (pouze jeden přechod označený žlutou barvou).



(a) (b)

Obrázek 22: Detekční úsečky. (a) otevřená dlaň, (b) zavřená dlaň.

Detekční mřížka s úsečkami a výstup ze senzoru při detekci dlaně je znázorněn na obrázku 23. Jako úsečky jsou použity vodorovné a svislé v jednotné mřížce, úsečky s procházející těžištěm detekčního čtverce postupně s otočením o pevně daný úhel a nakonec ještě několik úseček náhodně rozmístěných po obdélníku. Pro výpočet pozic úseček je použit celočíselný Bresenhamův algoritmus pro kreslení čar [19], který je implementován ve funkci `GetLinePoints` ve statické třídě `Bresenham`.



Obrázek 23: Konvexní detekce stavu dlaně.

Tento způsob detekce stavu dlaně funguje spolehlivě, pokud je osoba dostatečně blízko senzoru, hodí se tedy spíše pro situace, kdy je senzor využíván v sedícím režimu a je aktivně sledována pouze horní polovina těla. Senzorem poskytovaný *Depth stream* má nízké rozlišení a pokud je osoba více vzdálena od senzoru, senzor už vrací data s větším šumem [14], který na malém rozlišení má negativní vliv na získaná data. Rozměr výstupního obrazu je příliš malý, a proto detekce stavu není spolehlivá.

Popsaná detekce stavu, rozlišujícího jestli je dlaň otevřená nebo zavřená pomocí konvexnosti, je implementována třídou `ConvexityClosedHandTracker`.

Setrvání dlaně na místě

Další způsob detekce stavu dlaně je pomocí setrvání dlaně na místě. Minimální doba, po kterou musí dlaň zůstat na místě, je nastavitelná v konstruktoru nebo pomocí vlastnosti `MinimalHoverDuration`. Animace, která se má zobrazit během čekání, je nastavitelná pomocí metody `SetAnimationTexture`, ve výchozím stavu se žádná animace nezobrazuje.

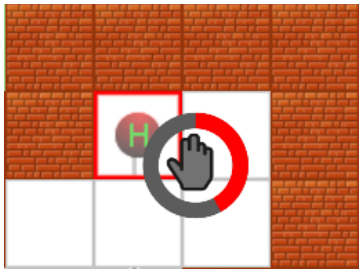
Tento způsob detekce setrvání na místě kontroluje pouze podle pozice kurzoru na obrazovce a nezohledňuje okolní kontext, například pro potvrzení tlačítka se pro setrvání může využít celá plocha tlačítka.

Popsaná detekce je implementována třídou `HoverHandTracker`.

6.4.2. Kurzor s animací

Kurzor s animací pomáhá zohlednit okolní kontext při potvrzování volby. Upravený kurzor je implementován třídou `KinectCircleCursor`, která dědí od základní komponenty `KinectCursor`. Komponenta `KinectCircleCursor` ponechává

rozhodování o tom, zda kurzor opustil nebo neopustil sledovanou oblast, na ostatních komponentách. Ty pomocí vlastnosti `Progress`, s rozsahem akceptovaných hodnot je z intervalu $[0, 1]$, mohou nastavit v jakém stavu je kontrola pozice dlaně. Pokud je hodnota větší než 0, je zobrazena kružnice okolo základního kurzoru, na které je graficky znázorněna aktuální hodnota. Jak tato kružnice vypadá je znázorněno na obrázku 24.



Obrázek 24: Animace kurzoru.

6.4.3. Tlačítko

V knihovně *Mach.Xna.Kinect* jsou k dispozici upravené verze tlačítek `KinectButton` a `KinectSpriteButton` rozšiřující základní komponenty pro tlačítka `Button` a `SpriteButton` z knihovny *Mach.Xna* o podporu vstupu ze senzoru *Kinect*.

Obě komponenty implementují rozhraní `IKinectAwareButton`. Toto rozhraní vyžaduje implementaci vlastnosti `MinimalHoverDuration`, která určuje minimální dobu, po kterou musí kurzor ovládaný senzorem *Kinect* zůstat nad daným tlačítkem, aby se považovalo tlačítko za potvrzené a vyvolala se událost `Selected`.

Zbylá funkcionalita tlačítek je identická s nadřazenou třídou `Button` resp. `SpriteButton`.

6.4.4. Dialogové okno

Varianta dialogového okna s podporou pro Kinect `KinectMessageBox` vychází z abstraktní třídy `MessageBoxBase`, akorát pro tlačítka je použita komponenta `KinectButton`.

6.4.5. Rozpoznávání gest

Komponenta pro rozpoznávání gest `Gestures` obsahuje pouze logiku, proto dědí od třídy `GameComponent`. K rozpoznávání gest využívá algoritmus DTW implementovaný knihovnou *Mach.Kinect*, podrobněji popsaný v kapitole 6.2.2.

Při vytváření nové instance komponenty je třeba v konstruktoru předat cestu k souboru s definovanými gesty případně rovnou předat gesta, která se mají rozpoznávat.

Po spuštění komponenty je při volání `Update` získána aktuálně sledovaná postava a zařazena do fronty ke zpracování. Jakmile fronta postav ke zpracování dosáhne požadované délky je v novém vlákně spuštěno rozpoznávání gest nad aktuálním obsahem fronty. Získané postavy jsou i během probíhajícího rozpoznávání stále vkládány do fronty. Ale pro spuštění dalšího rozpoznávání se čeká, dokud předchozí neskončí. Délka fronty je nastavitelná vlastností `MinimalFramesToProcess`.

Rozpoznaná gesta jsou postupně vkládána do druhé fronty, jejíž obsah lze získat metodou `GetRecognizedGestures`. Tato metoda vrací seznam rozpoznaných gest bez duplicit. Zároveň gesta odstraňuje z fronty, aby při dalším volání už nebyly znovu rozpoznány.

7. Implementace hry a pomocných aplikací

Tato kapitola se věnuje implementaci samotné hry *Atomix*, aplikace pro editaci úrovní hry a aplikace pro správu pohybových gest.

7.1. Logika hry

Projekt s logikou hry obsahuje třídy pro práci s herními úrovněmi a pro správu stavu rozehrané hry. Výstupem tohoto projektu je knihovna.

7.1.1. Definice úrovní

Do hry se herní úrovně nahrávají jako jeden definiční soubor. Třídy v tomto projektu s logikou se starají o vytvoření a zpracování tohoto definičního souboru. Definice všech úrovní je uložena ve třídě `GameDefinition`, která je serializovatelná. Serializací instance této třídy pomocí třídy `XmlSerializer` vznikne výsledný definiční soubor úrovní, který se používá ve hře. Jednotlivé úrovně jsou uloženy v poli přístupném přes vlastnost `Levels`. Úrovně v poli jsou seřazené v pořadí, ve kterém se mají hrát.

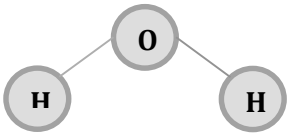
Každá úroveň je reprezentována třídou `Level` obsahující definici molekuly, která má být složena, rozložení dlaždic hracího pole a grafické podklady pro vykreslení jednotlivých dlaždic hracího pole.

Rozložení hracího pole i definice molekuly jsou implementovány stejně, jelikož definice molekuly je speciální případ rozložení hracího pole, které obsahuje pouze atomy ve správném rozložení.

Grafické podklady

Pokud je na hrací ploše umístěna stejná molekula vícekrát, ale liší se ve směru vazby, tak v grafických podkladech bude tato molekula uložena také vícekrát, pro každý směr vazby jednou. Díky tomu hra nemusí při každém překreslení obrazovky vytvářet pro atomy jejich vazby, stačí je načíst z definice. Kromě toho bude v definici vždy uložen i původní grafický podklad atomu bez žádné vykreslené vazby, i kdyby se na hrací ploše vyskytoval pouze s nějakou vazbou. Díky tomu bude možné později v editoru úrovní molekulu editovat, změnit typ nebo směr vazby.

Například pro molekulu vody na obrázku 25 bude atom vodíku uložen třikrát. Jednou pro vazbu směřující doleva, podruhé pro vazbu směřující doprava a potřetí bez žádné vazby. Atom kyslíku pouze dvakrát, jednou s vazbami a podruhé bez vazeb.



Obrázek 25: Různé směry vazby u vodíku.

Grafický obsah úrovně je reprezentován jako seznam podkladů dostupný přes vlastnost `Assets` na třídě `Level`.

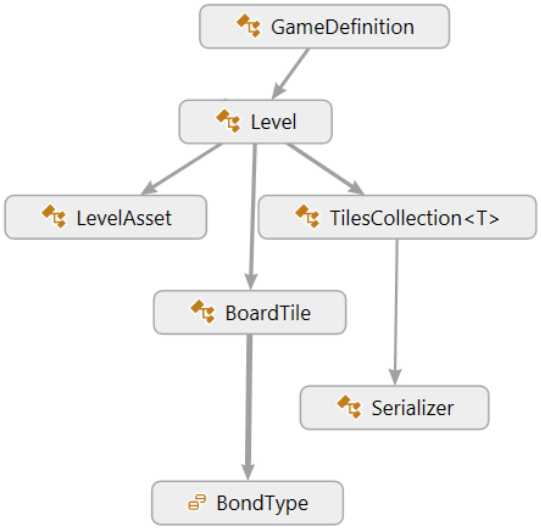
Každý grafický podklad je reprezentován instancí třídy `LevelAsset`, a je u něj nastavitelné jméno podkladu – vlastnost `Name`, kód, kterým lze na podklad odkazovat z hrací plochy – vlastnost `Code`, a samotným podkladem, který je ve formátu PNG a v base64 kódování – vlastnost `Content`.

Hrací plocha

Hrací plocha je dvourozměrná mřížka, ve které každá dlaždice reprezentovaná třídou `BoardTile` může obsahovat grafický podklad – vlastnost `Asset`. Pomocí hodnoty této vlastnosti je následně vyhledán odpovídající grafický podklad `LevelAsset`, podle vlastnosti `Code` v seznamu `Assets` v třídě `Level`. Je-li její pozice zafixovaná a nejde s ní po ploše hýbat – vlastnost `IsFixed`, je-li je volná – vlastnost `IsEmpty` – a tedy na její místo může být umístěna dlaždice atomu a údaj o násobnosti vazby pro každý směr.

Hrací mřížka je reprezentována kolekcí `TilesCollection`. Interně je hrací mřížka implementována kvůli serializovatelnosti jako jednorozměrné pole, ve kterém jsou jednotlivé řádky uloženy za sebou, ale pro pohodlnější práci je na kolekci implementován dvourozměrný indexer, kde první hodnota je index řádku a druhá hodnota index sloupce. Kolekce podporuje přidávání resp. odebírání řádků i sloupců pomocí metod `InsertRow`, `InsertColumn` resp. `RemoveRow`, `RemoveColumn`.

Pokud daná pozice mřížky nemá být pro hru vůbec využita, místo instance třídy `BoardTile` bude na jejím místě v kolekci přiřazena hodnota `null`.



Obrázek 26: Diagram závislostí u definice hry.

Diagram tříd použitých pro definici hry včetně jejich závislostí je na obrázku 26.

7.1.2. Stav hry

Stav hry si pamatuje, jaká úroveň byla naposledy hrána, které úrovně byly dohrány a s jakým výsledkem. Z výsledků se ukládá doba, za jakou byla molekula složena a kolik tahů po hrací ploše bylo pro její složení potřeba vykonat. Za lepší skóre se považuje takové, kde je potřeba méně tahů pro složení. Pokud je počet tahů stejný, jako u už existujícího nejlepšího skóre, pak se použije rychlejší čas. Přednost tedy má počet tahů nad dobou složení, jelikož při použití pohybového senzoru *Kinect* není čas rozhodující.

Stav hry je implementován třídou `GameState`, která je serializována při ukončení hry. Při novém spuštění hry je načtena definice úrovní, spočítá se její kontrolní součet a porovná se s uloženým kontrolním součtem v serializovaném stavu hry. Pokud si odpovídá, tak je stav hry použit i v této instanci hry. Pokud kontrolní součet nesedí, je uložený stav hry smazán a založen znovu, předpokládá se, že došlo ke změně úrovní.

Nejlepší výsledky pro úrovně hry spravuje třída `Highscore`, nejlepší výsledek dané úrovně spravuje třída `LevelHighscore`. Jednotlivé instance třídy `LevelHighscore` pro každou úroveň jsou poté uloženy v poli ve stejném pořadí, v jakém jsou seřazeny jednotlivé úrovně v definici hry. Aktualizaci nejlepšího výsledku pro danou úroveň implementuje funkce `UpdateIfBetter` na třídě `LevelHighscore`.

7.2. Hra

Samotná hra *Atomix* je implementována v herní knihovně XNA Framework a využívá ostatních komponent implementovaných v rámci knihoven popsaných v kapitole 6.

Hra je rozdělena na čtyři herní obrazovky:

`StartScreen`	úvodní obrazovka po spuštění hry
`LevelScreen`	obrazovka pro aktuálně hranou úroveň
`LevelsScreen`	přehled všech úrovní, které hra nabízí
`ErrorScreen`	chybová obrazovka, která je zobrazena pokud během hry dojde k problému s definičním souborem úrovní

Pro práci s obrazovkami je použit správce obrazovek z knihovny *Mach.Xna*, který je popsán v kapitole 6.3.4.

7.2.1. Inicializace hry

Během inicializace hry jsou připraveny společné komponenty, které jsou následně dostupné pro všechny herní obrazovky, je načtena definice úrovní hry a obnoven stav hry, pokud existuje.

Hra podporuje načítání uživatelských úrovní vytvořených pomocí editoru úrovní. Při každém spuštění hry je nejdříve zkontrolováno uživatelské úložiště, ve kterém se hledá soubor obsahující uživatelem definované úrovně. Pokud neexistuje, je načtena výchozí sada úrovní, která je součástí hry. Na rozlišení úložišť je využit XNA Framework a jeho koncept oddělených úložišť pomocí `StorageDevice`. Načítání úrovní implementuje třída `GameDefinitionFactory`.

V případě XNA projektu pro Windows je uživatelské úložiště reprezentováno složkou `SavedGames` umístěnou v Dokumentech aktuálně přihlášeného uživatele.

Po načtení úrovní je načten, pokud existuje, zaznamenaný stav z minulého spuštění hry. Je zkontrolováno, zda odpovídá kontrolní součet aktuálně načtené sady úrovní s kontrolním součtem uloženým v zaznamenaném stavu hry. Pokud neodpovídá je zaznamenaný stav smazán – je použit stav jako při prvním spuštění.

7.2.2. Úroveň hry

Při načtení úrovně hry na obrazovku je z definice reprezentované modelem `Level` vytvořena prezentační logika `LevelViewModel`, která k definici přidává logiku používanou při implementaci hry. Stejně tak pro jednotlivé dlaždice herní plochy je vytvořena prezentační logika `BoardTileViewModel`, která přidává dodatečné vlastnosti využívané pro vykreslování hry.

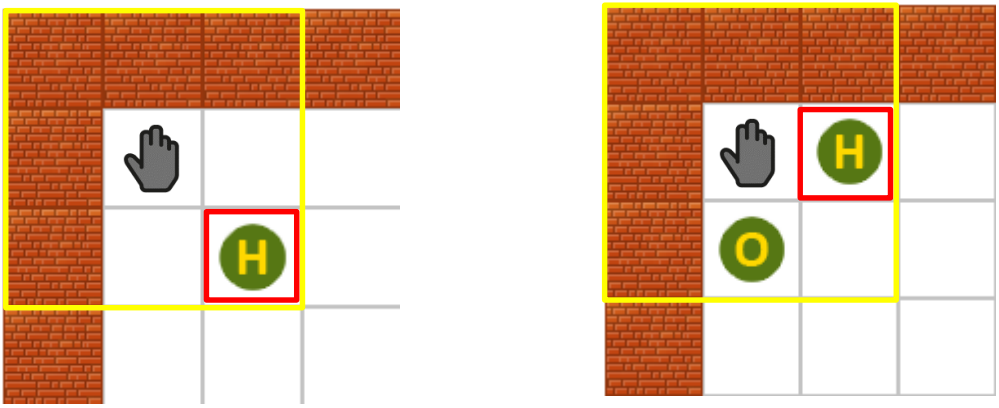
Při načítání obsahu jsou zároveň vypočítány souřadnice jednotlivých dlaždic na obrazovce metodou `CalculateBoardTilePositions`. Jejich pozice se během hry

nemění, při provedení tahu se pouze prohazují obsahy dlaždic. Tudíž stačí pozice vypočítat jednou. Stejná funkce se používá pro hrací plochu i pro vzorovou molekulu.

Při ovládání se rozlišuje, je-li je hra ovládána myší nebo senzorem *Kinect*. V případě ovládání pohybem pomocí senzoru je nastavena afinita výběru aktivní dlaždice k atomům. Pokud se pohybový kurzor přiblíží k dlaždici atomu na hrací ploše, je dlaždice s atomem označena za aktivní. Předpokládá se, že uživatel plánuje vybrat dlaždici s atomem pro provedení tahu hry.

Afinita pohybového kurzoru funguje tak, že pokud je kurzor přímo nad dlaždicí s atomem, je tato dlaždice označena za aktivní. Pokud je kurzor nad dlaždicí, která atom neobsahuje, je zkontrolováno bezprostřední okolí této dlaždice, a pokud se v tomto okolí vyskytuje dlaždice s molekulou, je tato dlaždice s atomem označena za aktivní, místo té nad kterou právě je pohybový kurzor. Pokud se v hledaném okolí vyskytuje více dlaždic s atomem, je zvolen jako aktivní ten, který je při hledání nalezen jako první. Hledání v okolí probíhá v pořadí z levého horního rohu do pravého dolního rohu. Afinita je implementována metodou `GetNeigbourMoleculeTile`.

Příklad určení aktivní dlaždice (označené červeným rámečkem) je na obrázku 27.



(a) (b)

Obrázek 27: Afinita pohybového kurzoru k atomům: (a) s jedním kandidátem; (b) s více kandidáty.

Afinita dlaždic urychluje ovládání hry pomocí senzoru *Kinect*, jelikož nevyžaduje po uživateli přesné pohyby, které s pomocí senzoru nejsou úplně snadně docílitelné.

Po vybrání atomu jsou pomocí metody `PrepareAvailableTileMovements` upraveny okolní dlaždice – pokud se atom smí v daném směru dlaždice pohnout, na dlaždici se zobrazí šipka odpovídající směru pohybu.

Pro posun dlaždice v daném směru pomocí senzoru se používá třída pro detekci gest posunu z knihovny *Mach.Kinect* popsaná v kapitole 6.2.2. Detekce gesta začíná

v okamžiku vybrání atomu se startovací pozicí zaznamenanou pohybovým kurzorem. Pokud dlaň vykoná potřebný posun v nějakém směru, je ověřeno, jestli tento směr je validní pro aktuální pozici atomu. Pro posun dlaždice myší stačí kliknout na odpovídající sousední dlaždici.

Když je vyvolán posun atomu ve zvoleném směru, metodou `GetNewAtomPosition` je zjištěno cílové místo, kam se má atom posunout. Tato pozice je následně uložena do pomocné proměnné a je spuštěna animace posouvající atom v tomto směru. Dokud není dokončena tato animace posunu, je ovládání hry pozastaveno.

Po každém odehraném tahu je pomocí metody `CheckFinish` zkontrolována hrací plocha, jestli nedošlo k sestavení hledané molekuly. Tato metoda projde aktuální rozestavění hrací plochy a pro každou buňku hrací plochy zkusí spárovat rozestavění vzorové molekuly. Pokud je zjištěna složená molekula na hrací ploše, je zobrazeno hlášení o úspěšném dokončení úrovně a nabídnuto pokračování další úrovní, případně opakování stejné úrovně znovu.

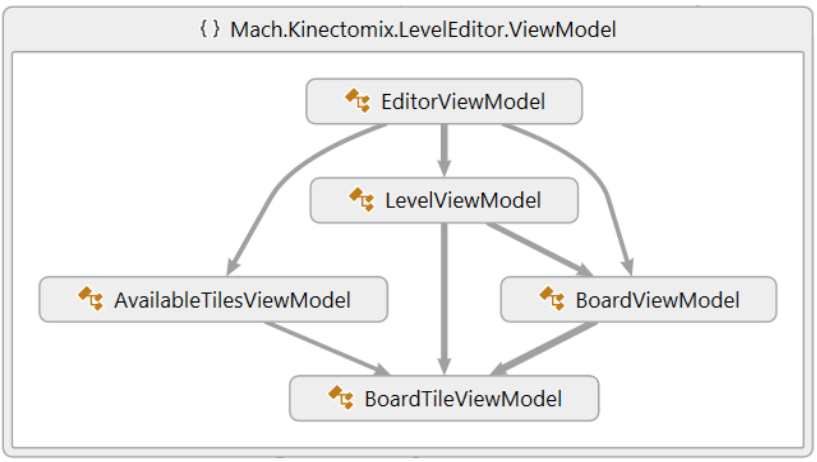
V obou případech je vytvořena nová instance obrazovky `LevelScreen` s definicí zvolené úrovně, aby další hraná úroveň začínala ve startovací pozici.

Hra nijak nekontroluje, jestli se rozestavěním atomů po hrací ploše uživatel dostal do situace, ze které hra nelze dokončit. Pokud uživatel tento stav zjistí, pomocí menu pauzy, musí aktuálně hranou úroveň restartovat ručně.

7.3. Aplikace pro editaci úrovní

Editor úrovní hry je vytvořen pomocí grafické knihovny WPF. Využívá knihovny *Mach.Wpf.Mvvm* popsané v kapitole 5.3 pro usnadnění práce s návrhovým vzorem MVVM. Dále využívá knihovnu s logikou hry *Mach.Kinectomix.Logic*, která je popsána v kapitole 7.1.

Hlavní prezentační logika editoru je umístěna ve třídě `EditorViewModel`, která spravuje celý životní cyklus editace úrovní. Pro definici jedné úrovně je prezentační logika ve třídě `LevelViewModel`, pro hrací plochu a molekulu slouží třída `BoardViewModel`, pro jednotlivé dlaždice hrací plochy je třída `BoardTileViewModel`. Prezentační logika ve třídě `AvailableTilesViewModel` udržuje informace o tom, jaké dlaždice smí být v editoru kresleny, rozlišuje se kontext hrací plochy a molekuly. Závislosti tříd prezentační logiky znázorňuje diagram na obrázku 28.



Obrázek 28: Diagram závislostí prezentační logiky.

Editor podporuje detekci toho, zda se od otevření existující definice provedla nějaká změna a je nutné definici znovu uložit. Jestli k nějaké změně došlo lze zjistit z vlastnosti `IsAnyPendingChange` na třídě `EditorViewModel`. Změny jsou detekovány tak, že třída sama se zapíše k události `PropertyChanged` a pokud je událost alespoň jednou vyvolána, došlo k nějaké změně.

Pro zobrazení hrací plochy je implementována upravená kolekce dlaždic `ObservableTilesCollection<T>`, která vychází z kolekce `TilesCollection<T>` v knihovně s logikou hry. Navíc implementuje rozhraní `INotifyCollectionChanged` a `INotifyPropertyChanged`, která jsou nutná pro správné fungování vazeb mezi grafickým rozhraním a prezentační logikou.

Kreslení dlaždic po hrací ploše je zajištěno pomocí třídy `TilePainter`, která implementuje koncept *Attached Behavior* [20] pomocí *Attached Properties*. Tato třída se registruje ke každému elementu dlaždice a zajišťuje překreslení na novou dlaždici nastavenou ve vlastnosti `PaintTile`.

Příklad použití třídy `TilePainter` k překreslení dlaždice je v ukázce 7.

```
<Image Source="{Binding AssetSource}"
       Behavior:TilePainter.IsPaintEnabled="True"
       Behavior:TilePainter.PaintTile="{Binding Path=PaintTile}"
```

Ukázka 7: Použití Attached Behavior.

Koncept *Attached Behavior* je použit také u třídy `TileBonds`, která rozhoduje o tom, jestli se u dané dlaždice mají zobrazit pomocné body pro nastavování vazeb atomů.

Samotné vykreslení těchto bodů zajišťuje třída `BondsAdorner` využívající koncept *Adorner*[18] pro kreslení grafických elementů mimo strom WPF elementů.

K vykreslení vazeb atomů na hrací ploše je využita třída `BondsVisualiser`, používající rozšiřující metodu `DrawBond` ke třídě `DrawingContext`, která je definována ve statické třídě v `DrawingContextExtensions`. Vazby jsou vykresleny ze středu plochy k okraji a nad takto vykreslené vazby je umístěn samotný atom tak, aby se navzájem obě tyto vrstvy překrývaly.

K načítání a ukládání definičního souboru slouží třída `LevelFactory`, která k práci se serializací definice využívá třídy `XmlSerializer`.

Pro změnu pořadí úrovní v seznamu pomocí přetahování je využito postupu, který je zveřejněn na stránkách společnosti Zag Studio[19].

Výstupem aplikace je definiční soubor obsahující všechny úrovně hry. Při ukládání definičního souboru je editorem automaticky přednastavena složka, kde hra očekává uživatelský definiční soubor. Pokud tato složka neexistuje, pokusí se ji editor založit.

7.4. Aplikace pro správu gest

Aplikace pro správu gest využívá knihovny *Mach.Wpf.Mvvm* popsané v kapitole 5.3. pro práci s návrhovým vzorem MVVM a knihovny *Mach.Kinect* pro práci s gesty.

Hlavní prezentační logika aplikace je umístěna do třídy `RecorderViewModel`. Prezentační logika jednoho gesta je ve třídě `GestureViewModel` a pro vybírání sledovaných bodů těla při zaznamenávání gesta je prezentační logika ve třídě `SkeletonViewModel`. Pro zobrazování zaznamenaného RGB videa a rozpoznaných postav je využito komponent `KinectStreamViewer` a `KinectSkeletonViewer` převzatých z Kinect SDK a upravených pro potřeby této aplikace.

Záznam nového gesta je spuštěn ve třídě `RecorderViewModel` metodou `StartRecordingCountdown`, která zahájí odpočet, aby se uživatel stihl připravit před senzor, a vytvoří novou instanci třídy `Recorder`, která je následně spuštěna metodou `Start`. Samotné předávání pozic bodů sledované postavy zaznamenávaného je v metodě `Sensor_SkeletonFrameReady`. V této metodě je také připravena logika pro rozpoznávání již zaznamenaných gest.

[18] http://msdn.microsoft.com/en-us/library/ms743737(v=vs.110).aspx (navštíveno 23. 7. 2014)
[19] http://www.zagstudio.com/blog/488 (navštíveno 22. 7. 2014)

Pro rozpoznávání zaznamenaných gest je využit algoritmus DTW implementovaný v knihovně *Mach.Kinect*. Jelikož jedno spuštění algoritmu je časově náročné je, stejně jako v XNA komponentě pro rozpoznávání gest, která je popsána v kapitole 6.4.5., i zde rozpoznávání spuštěno ve vedlejším vlákně, aby grafické rozhraní aplikace bylo responzivní i během rozpoznávání.

Jednotlivé zaznamenané pozice postav jsou ukládány do fronty. Když fronta dosáhne požadované délky je obsah fronty předán k rozpoznávání do vedlejšího vlákna. Během jeho běhu jsou pozice postav opět ukládány do fronty a nové rozpoznávání bude spuštěno, až to předcházející skončí. Na jedno spuštění může být rozpoznáno více gest.

Závěr

Během testování různých variant přirozeného uživatelského rozhraní hry byly zjištěny určité nedostatky stávající hardwarové verze senzoru *Kinect*, především v ohledu nepřesnosti získávaných dat z infračervené kamery. Postupným upravováním uživatelských gest se podařilo dosáhnout přirozeného uživatelského rozhraní, které je uživatelsky intuitivní a zároveň dostatečné použitelné i pro využití hry na veřejně přístupném místě, kde se často střídají různí uživatelé a nemají čas se dlouze zaobírat způsobem, jakým se hra ovládá.

Společnost *Microsoft* v červenci 2014 uvedla[20] na trh druhou verzi senzoru *Kinect*, která řeší určité nedostatky, které byly během implementace na první verzi senzoru zjištěny. Především vylepšuje a upřesňuje měření vzdálenosti díky vylepšené infračervené kameře, která je schopna zaznamenávat scénu ve vyšším rozlišení. Díky tomu by mělo být možné použít navrhovanou metodu detekce otevřené nebo zavřené dlaně i ve větší vzdálenosti od senzoru a nemusí se omezovat použití této metody pouze na režim v sedě. Zároveň vyšší rozlišení umožňuje lepší a stabilnější detekci sledovaných bodů postavy, a proto by pohyb kurzorem měl být plynulejší.

Během implementace hry *Atomix* byl výsledný projekt se zdrojovým kódem rozdělen na spustitelné aplikace a tři samostatné knihovny, jejichž kód je na hře nezávislý, a ostatní vývojáři tyto knihovny mohou využít jako základ při programování svých vlastních aplikací využívající senzor *Kinect* nebo herní platformu XNA Framework.

Knihovna *Mach.Kinect* obsahuje funkcionalitu pro práci se senzorem *Kinect* a není nijak závislá na platformě XNA Framework. Knihovna *Mach.Xna* obsahuje komponenty pro platformu XNA Framework, které nejsou závislé na přítomnosti senzoru *Kinect*. Poslední knihovna *Mach.Xna.Kinect* rozšiřuje základní knihovnu pro XNA Framework *Mach.Xna* o komponenty využívající senzor *Kinect*.

Knihovna *Mach.Kinect* s implementací rozpoznávání obecných gest nemusí být využívána jen při tvorbě počítačových her, ale mohla by posloužit například i jako základ pro rozpoznávání znakové řeči. K tomu by bylo ovšem potřeba vytvořit dostatečně obsáhlou sadu referenčních gest pro jednotlivé znaky.

I přes to, že pro implementaci byl původně použit originální XNA Framework, všechny knihovny vytvořené za použití platformy XNA Framework se podařilo zkompilovat

[20] http://blogs.msdn.com/b/kinectforwindows/archive/2014/07/15/the-kinect-for-windows-v2-sensor-and-free-sdk-preview-are-here.aspx (navštíveno 20. 7. 2014)

i při použití alternativní platformy MonoGame a součástí implementace je i sada knihoven využívající právě tuto platformu.

Ačkoliv XNA Framework podporuje vývoj i pro herní konzole XBOX 360, není zatím k dispozici veřejně dostupná knihovna pro práci se senzorem Kinect na této platformě. Z tohoto důvodu je stávající implementace hry funkční pouze na platformě Windows.

Použitá literatura

[1] Thalion Software GmbH, Atomix Game Manual, 1990.

[2] D. Catuhe, Programming with the Kinect for Windows Software Development Kit, 2012.

[3] Microsoft, „Kinect for Windows Programming Guide - Data Streams," [Online]. Available: http://msdn.microsoft.com/en-us/library/hh973075.aspx. [Přístup získán 28. 7. 2014].

[4] Microsoft, Human Interface Guidelines v1.8, 2013.

[5] K. Khoshelham, „Accuracy analysis of Kinect Depth data," v *ISPRS Calgary 2011 Workshop*, 2011.

[6] B. Freedman, A. Shpunt, M. Machline a Y. Arieli, „Depth mapping using projected patterns". Patent US 2010/0118123 A1, 13 Květen 2010.

[7] K. Khoshelham a S. O. Elberink, „Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications," v *Sensors*, 2012.

[8] C. Sinthanayothin, N. Wongwaen a W. Bholsithi, „Skeleton Tracking using Kinect Sensor & Displaying in 3D Virtual Scene," v *International Journal of Advancements in Computing Technology (IJACT)*, 2012.

[9] M. Azimi, „Skeletal Joint Smoothing White Paper," 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131429.aspx. [Přístup získán 10. 7. 2014].

[10] G. Li, Y. Wang, M. Li a Z. Wu, „Similarity Match in Time Series Streams under Dynamic Time Warping Distance," v *Computer Science and Software Engineering, 2008 International Conference*, 2008.

[11] P. Senin, „Dynamic Time Warping algorithm Review," University of Hawaii at Manoa, Honolulu, USA, 2008.

[12] M. Müller, Information Retrieval for Music and Motion, 2007.

[13] M. Raptis, D. Kirovski a H. Hoppe, „Real-Time Classification of Dance Gestures from Skeleton Animation," v *Symposium on Computer Animation*, 2011.

[14] K. Khoshelham, „Accuracy analysis of Kinect depth data," v *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2011.

[15] A. Reed, Learning XNA 4.0: Game Development for the PC, Xbox 360, and Windows Phone 7, O`REILLY, 2010.

[16] J. Smith, „WPF Apps With The Model-View-ViewModel Design Pattern," Microsoft, 2009. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx. [Přístup získán 22. 7. 2014].

[17] R. Garofalo, Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern, O'Reilly Media, Inc., 2011.

[18] G. Simion, V. Gui a M. Otesteanu, „Finger Detection Based on Hand Contour and Colour Information," v *Applied Computational Intelligence and Informatics (SACI), 2011 6th IEEE International Symposium*, 2011.

[19] J. E. Bresenham, „Algorithm for computer control of a digital plotter," v *IBM Systems Journal*, 1965.

[20] A. Nathan, WPF 4.5 Unleashed, 2013.

Příloha A. Obsah přiloženého DVD-ROM

Součástí této diplomové práce je disk DVD-ROM s následujícím obsahem:

`/DP.pdf`	text této diplomové práce
`/Install/Kinectomix/Game/`	adresář obsahující instalátor hry
`/Install/Kinectomix/Editor/`	adresář obsahující instalátor editoru úrovní
`/Install/GestureRecorder/`	adresář obsahující instalátor aplikace pro správu gest
`/Documentation/Kinectomix.pdf`	uživatelská dokumentace ke hře
`/Documentation/Gestures.pdf`	uživatelská dokumentace aplikace pro nahrávání gest
`/Documentation/Reference/`	adresář obsahující zkompilovanou referenční dokumentací knihoven a logiky hry
`/Redist/`	adresář s aplikacemi a knihovnami třetích stran potřebnými k spuštění aplikací
`/SDK/`	adresář s SDK knihovnami potřebnými pro vývoj a zkompilování zdrojových kódů
`/Source/Kinectomix/`	zdrojový kód implementace hry a editoru úrovní
`/Source/Library/`	zdrojový kód knihoven
`/Source/GestureRecorder/`	zdrojový kód aplikace pro správu gest
`/Binaries/Xna/`	knihovny zkompilované pomocí XNA Framework
`/Binaries/MonoGame/`	knihovny zkompilované pomocí MonoGame
`/Samples/`	ukázková zaznamenaná gesta a definice úrovní

Příloha B. Použité zkratky a standardy

API	Programovatelné aplikační rozhraní
DMO	DirectX Media Object
DTW	Dynamic Time Warping
FPS	Počet snímků za vteřinu
MVC	Model-View-Controler
MVVM	Model-View-ViewModel
NUI	Natural User Interface
PHIZ	Optimální interakční zóna
RGB	Barevný model obrazu červená-zelená-modrá
SDK	Software Developement Kit
WPF	Windows Presentation Foundation
XNA	Herní Framework

Příloha C

UŽIVATELSKÁ DOKUMENTACE

Hra Kinectomix

Tato uživatelská dokumentace je součástí
diplomové práce Atomix pro Microsoft Kinect pro Windows.

Vladimír Mach

2014

Obsah

1. Úvod

Tato uživatelská dokumentace je součástí diplomové práce *Atomix pro Microsoft Kinect pro Windows*.

Dokumentace popisuje v kapitole 2 ovládání hry *Kinectomix* a v kapitole 3 je popsán editor pro vytváření vlastních úrovní hry.

2. Hra Kinectomix

Hra *Kinectomix* je upravená verze hry *Atomix*. Hra využívá pohybového senzoru *Kinect* společnosti Microsoft a umožňuje ovládání hry pohybem. Uživatelské rozhraní hry je v českém a anglickém jazyce a přepíná se automaticky podle nastavení aktuálního počítače. Základní sada úrovní molekul odpovídá sadě v originální hře *Atomix* s vynecháním bonusových úrovní. Úrovně byly vytvořeny z podkladů projektu *kp-atomix*[1].

2.1. Požadavky na běh

Hra pro svůj běh vyžaduje .NET Framework 4.0, běhové prostředí XNA Framework 4.0 Refresh a běhové prostředí pro senzor Kinect. Běhová prostředí jsou k dispozici na přiloženém disku DVD ve složce `Redist`.

Pro správné fungování hry je třeba grafické karty podporující DirectX 9.0c a minimální rozlišení obrazovky alespoň 1280 × 720 bodů.

2.2. Instalace hry

Instalátor hry `setup.exe` je umístěn na přiloženém disku DVD ve složce `Install/Kinectomix/`.

Před instalací hry je potřeba nainstalovat běhové prostředí pro senzor Kinect pomocí instalátoru `KinectRuntime-v1.8-Setup.exe` umístěném v adresáři `Redist` a běhové prostředí XNA Framework pomocí instalátoru `xnafx40_redist.msi` také v adresáři `Redist` na přiloženém disku DVD.

Po jeho spuštění bude zobrazen jednoduchý průvodce, který provede procesem instalace. Po dokončení instalace bude na ploše a v nabídce start umístěn zástupce s názvem **Kinectomix**, kterým lze hru spustit (obrázek 1).

Obrázek 1: Ikona hry.

[1] https://kp-atomix.googlecode.com/hg/levels/original.json (navštíveno 20. 7. 2014)

2.3. Pravidla hry

Hrací plocha hry je dvourozměrná mřížka. Každá buňka mřížky je buď zeď, atom nebo volný prostor. Buňkami se zdí nelze po hrací ploše pohybovat. Atomy jsou různého druhu, lišící se typem atomu a vazbou. Po hrací ploše jimi lze pohybovat pouze vertikálně nebo horizontálně a to pouze k nejbližší překážce v daném směru.

Za překážku se považuje buď zeď, nebo jiný atom. Toto uspořádání je jedna úroveň hry, hra samotná sestává z více úrovní lišících se zadanou molekulou a uspořádáním buněk hrací plochy.

Úroveň je vyhrána, pokud jsou poskládány jednotlivé atomy po hrací ploše do molekuly, která byla zadána. Řešení, jak lze danou molekulu na hrací ploše poskládat, může existovat více, liší se například počtem nutných tahů pro složení hry.

Za lepší řešení se považuje takové, u kterého je potřeba méně tahů pro složení. Pokud je počet tahů stejný, jako u už existujícího nejlepšího skóre, pak se použije rychlejší čas. Přednost má tedy počet tahů nad dobou složení.

2.4. Ovládání

Hra se dá ovládat myší nebo, po připojení senzoru *Kinect*, pohybem.

Při ovládání pohybem je zobrazen vlastní kurzor, který je mapován na dlaň sledovaného hráče. Je sledována ta dlaň, která je blíže k senzoru *Kinect*. Aby dlaň vůbec sledována byla, je nutné ji mít alespoň 20 centimetrů od těla před sebou. Pro potvrzení volby je třeba dlaň podržet alespoň 2 vteřiny nad požadovaným objektem (tlačítko, atom apod.). Během čekání na potvrzení je zobrazena kružnice okolo kurzoru, která je postupně vyplňována. Pro zrušení potvrzování stačí přesunout dlaní kurzor mimo sledovaný objekt.

2.4.1. Po spuštění hry

Po spuštění je hra zobrazena v okně, pro přepnutí do režimu zobrazení na celou obrazovku slouží klávesová zkratka `F11`.

Pokud není připojen senzor *Kinect*, v horní části obrazovky se zobrazí upozornění vyzývající k připojení senzoru, viz obrázek 2. Po připojení senzoru se zde bude zobrazovat aktuální průběh inicializace senzoru.

Obrázek 2: Bez připojeného senzoru Kinect.

Po úspěšným připojení senzoru je toto upozornění skryto.

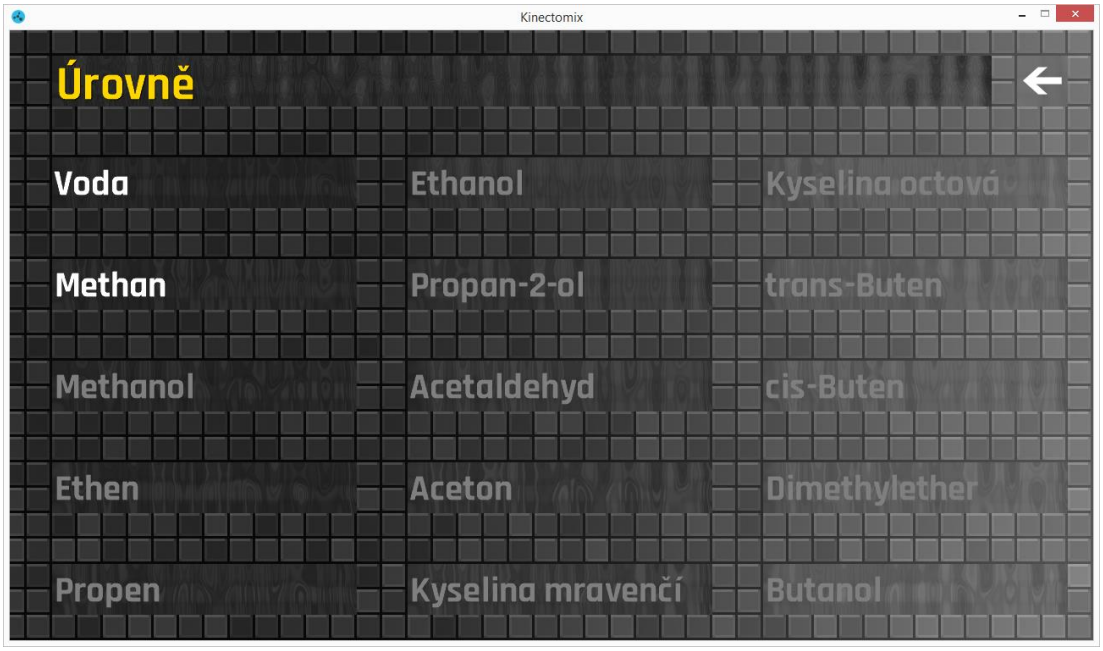
Obrázek 3: S připojeným senzorem Kinect.

Na hlavní obrazovce jsou k dispozici tři tlačítka:

hrát	spustí hru od úrovně, na které se naposledy skončilo
úrovně	seznam všech úrovní
ukončit hru	uloží aktuální stav hry a ukončí ji

2.4.2. Přehled úrovní

Obrazovka s přehledem úrovní umožňuje spustit libovolnou úroveň hry. Ovšem je možné spustit jen tu úroveň, která byla už dohrána nebo nejbližší následující po dohrané. Hratelné úrovně jsou vykresleny bílým písmem, šedým zatím neaktivní. Pro stránkování v seznamu úrovní lze využít ikon v pravém horním rohu, šipek na klávesnici nebo pohybovým gestem posunu doleva resp. doprava.



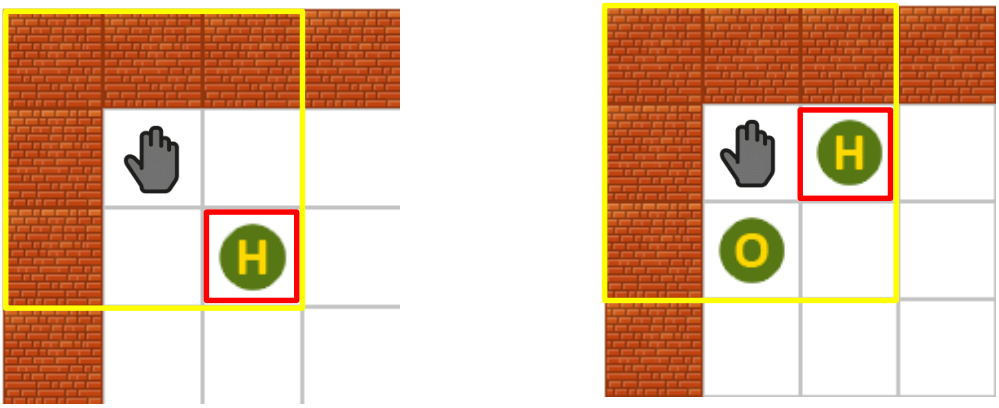
Obrázek 4: Přehled úrovní se dvěma hratelnými úrovněmi.

2.4.3. Úroveň

Po spuštění nové úrovně se zobrazí obrazovka, ve které je v levém sloupci jméno molekuly, molekula, kterou je nutné složit, a skóre spuštěné úrovně.

Při ovládání se rozlišuje, jestli je hra ovládána myší nebo senzorem *Kinect*. V případě myši je vždy aktivní ta dlaždice, nad kterou je umístěn systémový kurzor. V případě ovládání pohybem pomocí senzoru je nastavena afinita výběru aktivní dlaždice k atomům, a pokud se pohybový kurzor přiblíží k dlaždici atomu na hrací ploše, je označena za aktivní dlaždice s atomem.

Afinita pohybového kurzoru funguje tak, že pokud je kurzor přímo nad atomem, je tato dlaždice označena za aktivní. Pokud je kurzor nad dlaždicí, která atom neobsahuje, je zkontrolováno okolí této dlaždice. Pokud se v tomto okolí vyskytuje dlaždice s atomem, je tato dlaždice s atomem označena za aktivní, místo té, nad kterou je umístěn pohybový kurzor. Jestliže se v hledaném okolí vyskytuje více dlaždic s atomem, je zvolena jako aktivní ta, která je při hledání nalezena dříve. Prohledávání okolí probíhá v pořadí z levého horního rohu do pravého dolního rohu. Příklad afinity je zobrazen na obrázku 5.

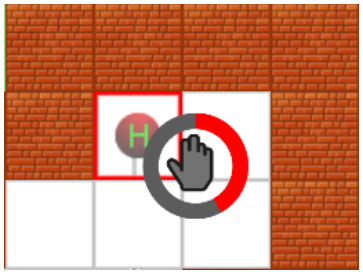


(a) (b)

Obrázek 5: Afinita pohybového kurzoru k atomům: (a) s jediným kandidátem; (b) s více kandidáty

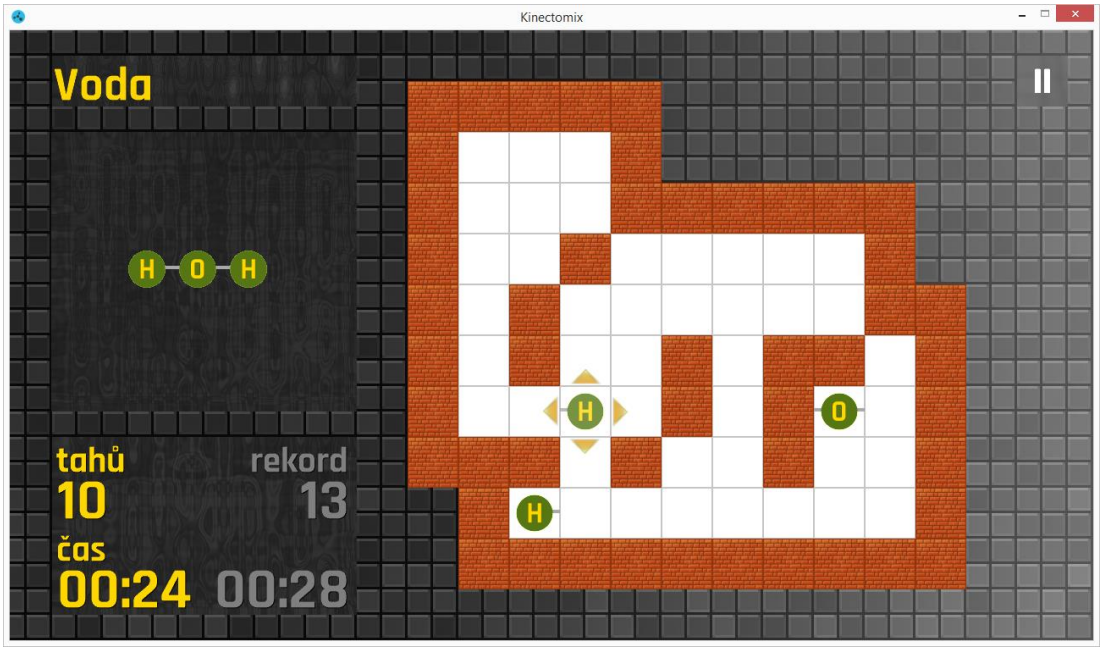
Tato afinita dlaždic usnadňuje ovládání hry díky větší toleranci na pozici dlaně.

Pro vybrání atomu na hrací ploše je třeba na něj kliknout levým tlačítkem myši respektive v případě ovládání pomocí senzoru *Kinect* podržet dlaň v blízkosti atomu, dokud nedoběhne potvrzovací animace okolo kurzoru. Potvrzovací animace je znázorněna na obrázku 6.



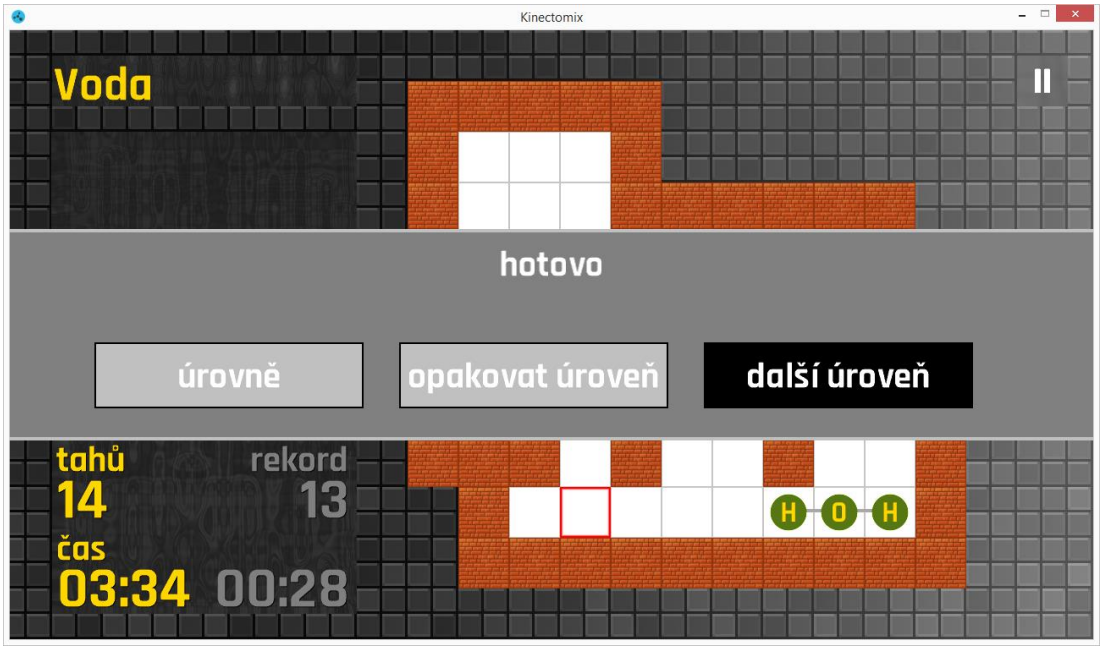
Obrázek 6: Potvrzovací animace kurzoru.

Po vybrání atomu se okolo něj zobrazí šipky ve směrech, ve kterých se atom smí po hrací ploše pohnout. Kliknutím myší na šipku, respektive pohybem dlaní ve směru šipky, se vybraný atom v tomto směru posune po hrací ploše. Ukázka vybraného atomu je na obrázku 7.



Obrázek 7: Spuštěná úroveň s vybraným atomem.

Po dokončení úrovně se zobrazí nabídka umožňující opakování stejné úrovně znovu, přejít na přehled všech úrovní nebo pokračovat další úrovní. Potvrzovací nabídka je zobrazena na obrázku 8.



Obrázek 8: Potvrzovací nabídka.

Pokud byla dohrána poslední úroveň hry, po potvrzení tlačítka *další úroveň* se místo další úrovně zobrazí hlavní nabídka hry.

2.5. Klávesové zkratky

Pro nastavení hry jsou k dispozici klávesové zkratky popsané v tabulce.

Zkratka	Vyvolaná akce
`F11`	Přepíná mezi režimem celé obrazovky a oknem.
`Q`	Natočí senzor *Kinect* o 5° vzhůru.
`A`	Natočí senzor Kinect o 5° dolů.

Tabulka 1: Přehled klávesových zkratek.

2.6. Uživatelské úrovně

Hra podporuje načítání vlastní sady úrovní vytvořených uživatelem.

Při spuštění hry jsou nejdříve vyhledány uživatelské úrovně uložené v souboru `Definition.atx`, který je očekáván v adresáři **Dokumenty** aktuálně přihlášeného uživatele v podsložce `SavedGames\Kinectomix\Game\AllPlayers`. Pokud tento soubor není nalezen, je načtena výchozí sada úrovní, která byla nainstalována společně s hrou.

Vlastní úrovně umožňuje vytvořit editor popsaný v kapitole 3 této dokumentace.

3. Editor úrovní

Editor úrovní umožnuje vytvořit vlastní sadu úrovní do hry. Výstupem aplikace je definiční soubor obsahující všechny úrovně hry.

3.1. Požadavky na běh

Editor úrovní vyžaduje .NET Framework 4.5. Požadovaný operační systém je Windows Vista SP2 a novější.

3.2. Instalace editoru

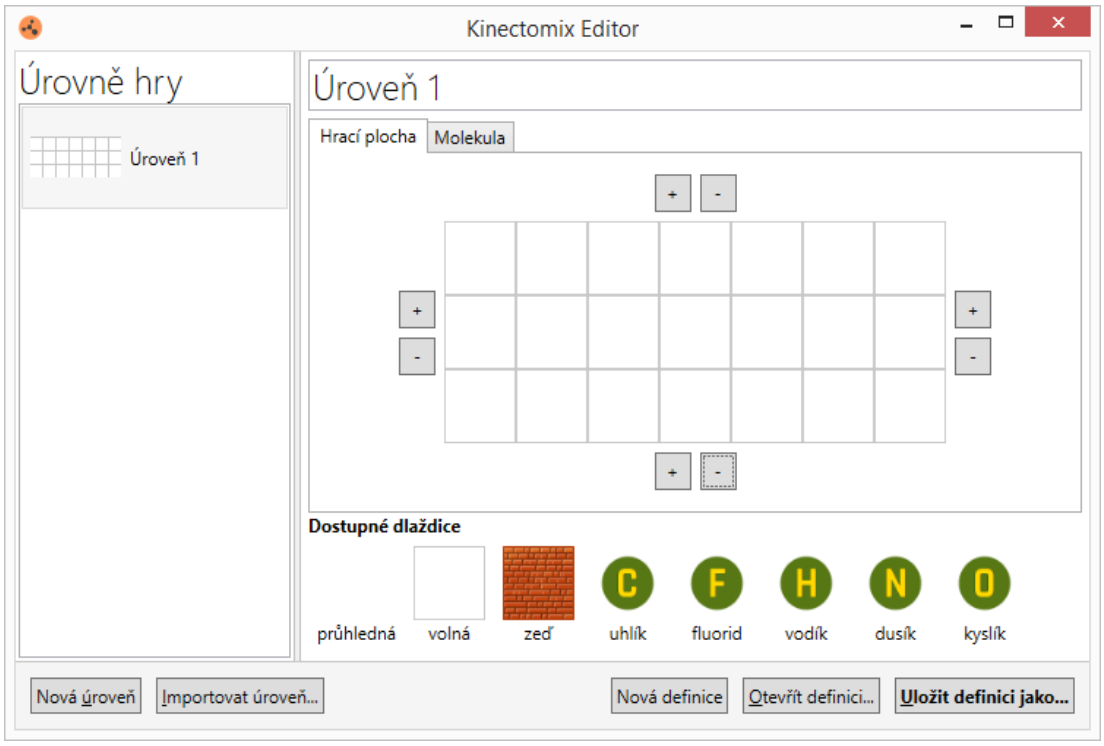
Instalátor editoru `setup.exe` je umístěn na přiloženém DVD ve složce `Install/Kinectomix.Editor/`. Po jeho spuštění bude zobrazen jednoduchý průvodce, který provede procesem instalace. Po dokončení instalace bude na ploše a v nabídce Start umístěn zástupce s názvem **Kinectomix Editor**, kterým lze hru spustit. Ikona aplikace je na obrázku 9.

Obrázek 9: Ikona editoru úrovní.

Při instalaci se také v systému zaregistruje přípona `.atx` pro použití editorem úrovní. K otevření definičního souboru v editoru stačí otevřít definiční soubor a editor se otevře rovnou k editaci tohoto souboru.

3.3. Rozvržení aplikace

Okno aplikace je rozděleno na tři oblasti. Levá část obrazovky obsahuje seznam všech úrovní obsažených v definici s jejich náhledy. Pravá část obrazovky umožňuje úpravu rozložení aktuálně vybrané úrovně v seznamu úrovní vlevo. V dolní části obrazovky jsou umístěna tlačítka určena pro manipulaci s úrovněmi.



Obrázek 10: Aplikace po spuštění.

3.4. Správa definic

Po spuštění aplikace se automaticky otevře okno editoru s novou definicí úrovní obsahující jednu prázdnou úroveň.

3.4.1. Nová definice

Nová definice obsahující jednu prázdnou úroveň se založí kliknutím na tlačítko **Nová definice**. Pokud jsou provedeny nějaké neuložené změny v aktuálně otevřené definici úrovní, aplikace na tuto skutečnost upozorní a umožní vytvoření nové definice zrušit.

3.4.2. Otevření existující definice

Pro načtení existující definice úrovní slouží tlačítko **Otevřít definici...**. Po kliknutí na něj se otevře dialogové okno pro výběr souboru obsahující definici úrovní. Jako výchozí složka se volí ta, kde hra *Kinectomix* hledá definice úrovní, pokud existuje. Pokud jsou provedeny nějaké neuložené změny v aktuálně otevřené definici úrovní, aplikace na tuto skutečnost upozorní a nabídne možnost otevírání jiné definice zrušit.

3.4.3. Uložení definice

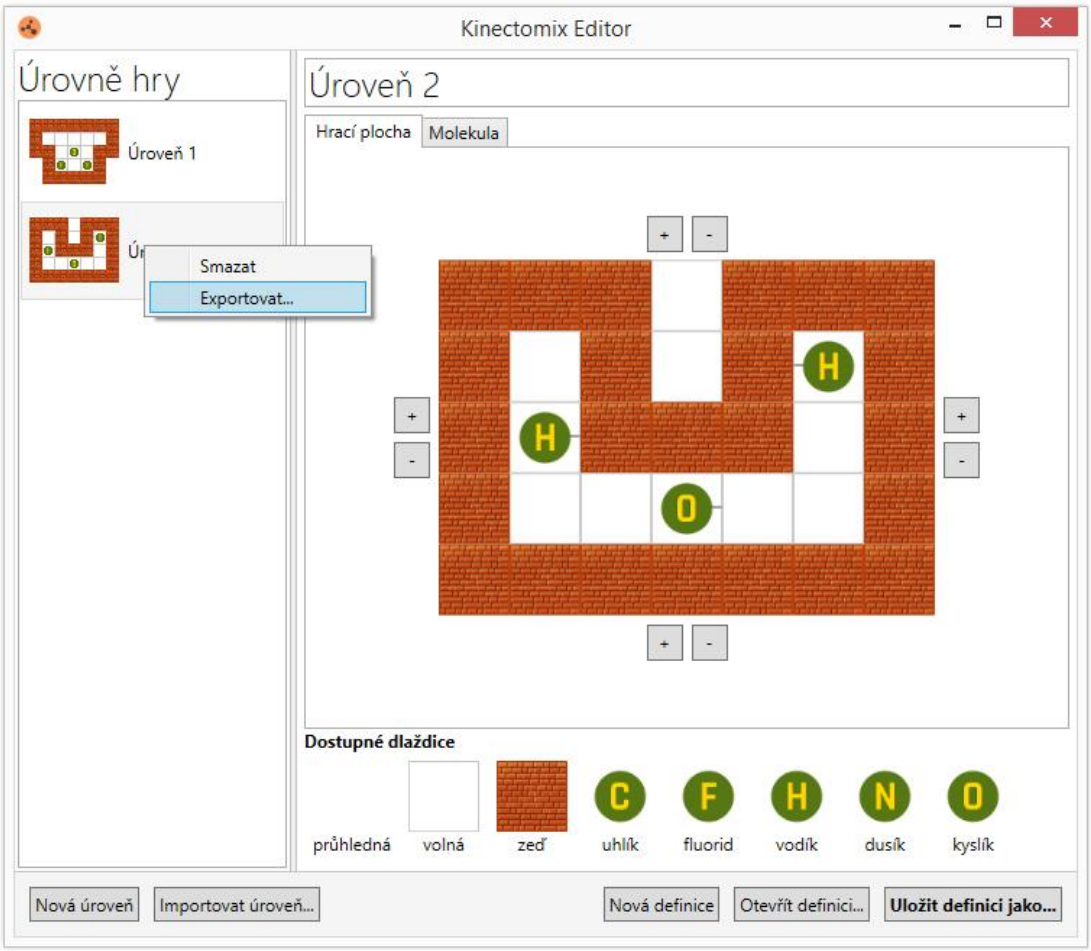
Pro uložení upravené definice úrovní slouží tlačítko Uložit definici jako..., které zobrazí dialogové okno pro výběr umístění, kam se má definice úrovní uložit. Výchozí složka je nastavena na složku, kde hra při spuštění hledá definiční soubor. Jedná se o složku

`SavedGames\Kinectomix\Game\AllPlayers` umístěnou v dokumentech aktuálně přihlášeného uživatele.

3.5. Správa úrovní

V levém sloupci aplikace je seznam všech úrovní, které aktuální definice obsahuje. Pro přidání nové úrovně do aktuální definice slouží tlačítko **Nová úroveň**, po kliknutí na něj se přidá nakonec seznamu úrovní prázdná úroveň. Pro odebrání úrovně z definice je třeba na definici v seznamu kliknout pravým tlačítkem a v kontextovém menu vybrat možnost **Smazat**. Kromě smazání kontextové menu také obsahuje možnost **Exportovat…**, které umožňuje vybranou úroveň exportovat do samostatného souboru, který se následně může pomocí tlačítka **Importovat úroveň…** přidat do definice. Tímto způsobem lze přesouvat již existující úrovně mezi různými definicemi úrovní.

Snímek obrazovky editoru s exportem úrovně je na obrázku 11.



Obrázek 11: Exportování vybrané úrovně.

3.6. Editace úrovně

V pravé části obrazovky je možnost editovat vybranou úroveň ze seznamu úrovní. Pro každou úroveň lze nastavit její název, rozložení hrací plochy a rozložení vzorové molekuly. Přepínání mezi rozložením hrací plochy a molekuly je pomocí záložek umístěných pod nadpisem. Molekula i hrací plocha se editují stejným způsobem. Liší se pouze v tom, jaké druhy dlaždic lze na plochu umístit, toto kontroluje aplikace při přepnutí záložek.

Velikost hrací plochy lze měnit pomocí tlačítek `+` a `-` umístěných po stranách hrací plochy.

Pro změnu dlaždice hrací plochy je nutné nejdříve vybrat, jaká dlaždice se má kreslit v sekci **Dostupné dlaždice** kliknutím na ni. Vybraný typ dlaždice je v seznamu dostupných zvýrazněn. Poté je možné tuto dlaždici kreslit po hrací ploše, buď jednotlivě kliknutím na dlaždici, nebo táhnutím se stisknutým levým tlačítkem po hrací ploše. Pro vypnutí kreslícího režimu stačí na danou dlaždici v seznamu dostupných znovu kliknout a odznačí se.

Typy dlaždic použitelných v editoru jsou uvedeny v tabulce 2.

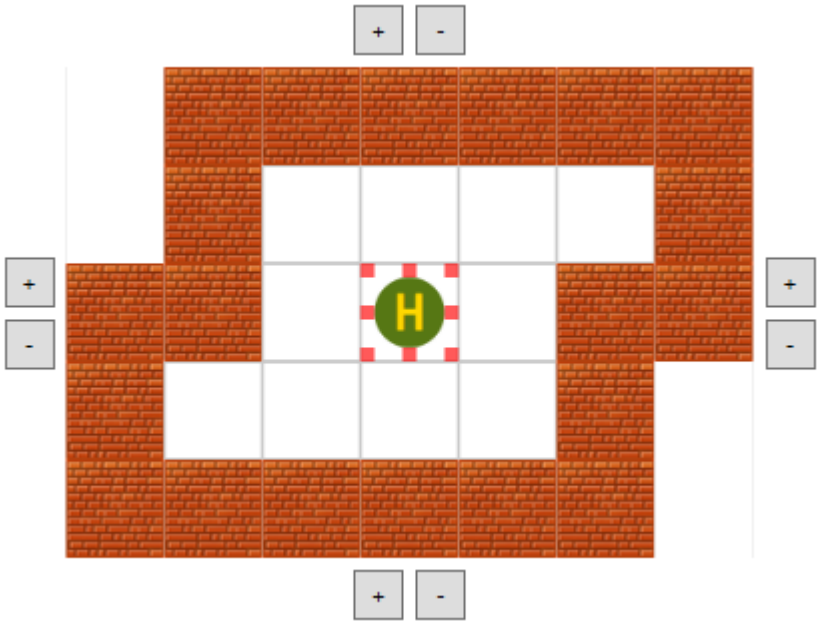
Typ dlaždice	Význam
`Průhledná`	Tato dlaždice nebude ve hře vůbec vykreslena.
`Volná`	Volná hrací plocha, po které se může pohybovat atom.
`Zeď`	Překážka pro atomy. Uživatelsky definovatelná dlaždice.
`Atom`	Pohyblivý atom. Uživatelsky definovatelná dlaždice.

Tabulka 2: Typy dlaždic.

Grafické podklady použité v úrovni jsou při uložení definice součástí uloženého souboru. Proto při načtení existující úrovně editorem jsou mezi dostupnými dlaždicemi zobrazeny jak ty výchozí, tak i ty načtené z úrovně. Podklady načtené z definičního souboru jsou označeny černým rámečkem. Na obrázku 12 jsou 4 grafické podklady načteny z definičního souboru.

Obrázek 12: Dostupné dlaždice načtené z definice.

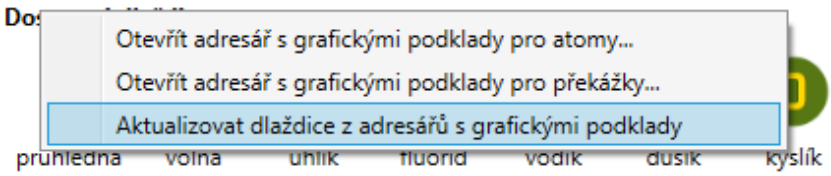
Pro nastavení vazeb atomu na hrací ploše je potřeba nejprve vypnout kreslící režim vypnutím vybrané dlaždice v seznamu dostupných dlaždic. Poté stačí najet myší na vybraný atom a zobrazí se 8 červených bodů ve směru nastavitelných vazeb, jak je vidět na obrázku 13. Kliknutím na bod se v daném směru bodu přidá vazba. Opakovaným kliknutím se zvyšuje vazebnost, dokud nedosáhne maximální hodnoty 6, při dalším kliknutí se vazby zruší úplně a pokračuje se zpět jako na začátku.



Obrázek 13: Nastavování vazeb atomu.

Editor nijak nekontroluje, jestli vytvořená molekula je platná, ani jestli si vazby navzájem odpovídají. Tato kontrola je ponechána na uživateli.

Kromě výchozí sady atomů a zdí lze do programu přidat vlastní grafické podklady. Aplikace je hledá ve složce `Tile\Atom` pro atomy a `Tile\Fixed` pro překážky. Obě tyto cesty jsou relativní vůči spustitelnému souboru editoru úrovní. Otevření odpovídajících složek lze v kontextovém menu po kliknutím pravým tlačítkem na nadpis Dostupné dlaždice. Po změně grafických podkladů je nutné je v aplikaci aktualizovat. Aktualizaci lze provést ve stejném kontextovém menu vybráním volby Aktualizovat dlaždice z adresářů s grafickými podklady. Snímek obrazovky této kontextové nabídky je na obrázku 14.



Obrázek 14: Kontextové menu pro úpravu grafických podkladů.

Příloha D

UŽIVATELSKÁ DOKUMENTACE

Aplikace pro záznam pohybových gest

Tato uživatelská dokumentace je součástí
diplomové práce Atomix pro Microsoft Kinect pro Windows.

Vladimír Mach

2014

Obsah

1. Úvod

Tato uživatelská dokumentace je součástí diplomové práce *Atomix pro Microsoft Kinect pro Windows*.

Dokumentace popisuje práci s aplikací pro záznam pohybových gest.

2. Správce gest

Aplikace pro správu gest usnadňuje zaznamenání a následné ověření gest za použití senzoru *Kinect*. Tato aplikace je určena pro vývojáře používající knihovnu *Mach.Kinect* k rozpoznávání obecných pohybových gest. Z tohoto důvodu je rozhraní aplikace pouze v anglickém jazyce.

2.1. Požadavky na běh

Hra pro svůj běh vyžaduje .NET Framework 4.0 a běhové prostředí pro Microsoft Kinect ve verzi 1.8.

2.2. Instalace aplikace

Instalátor aplikace `setup.exe` je umístěn na přiloženém DVD ve složce `Install/Kinect.GestureRecorder/`.

Před instalací aplikace je třeba nainstalovat běhové prostředí pro senzor Kinect pomocí instalátoru `KinectRuntime-v1.8-Setup.exe` umístěném v adresáři `Redist` na přiloženém disku DVD.

Po jeho spuštění bude zobrazen jednoduchý průvodce, který provede procesem instalace. Po dokončení instalace bude na ploše a v nabídce Start umístěn zástupce s názvem **Gesture Recorder**, kterým lze program spustit (obrázek 1).

Obrázek 1: Ikona aplikace.

Aplikace při instalaci také registruje příponu `.gst` pro soubory obsahující uložená gesta.

2.3. Struktura aplikace

Okno aplikace je rozděleno na tři části. Levá část je určena k nahrání nového gesta, pravá část pro ověření dříve nahraných gest. Prostřední část okna zobrazuje video z připojeného senzoru *Kinect* s rozpoznanými postavami a aktuální počet snímků za vteřinu získávaných ze senzoru. Uprostřed dolní části obrazovky je zobrazena cena rozpoznaného gesta. Tato hodnota slouží k ověření přesnosti rozpoznaného gesta oproti vzorovým gestům. Čím nižší hodnota tohoto čísla je, tím přesněji bylo gesto rozpoznáno.

Aplikace předpokládá, že v době záznamu gesta bude v dosahu senzoru pouze jedna osoba, jejíž pohyb bude zaznamenáván.

2.4. Formát zaznamenaného gesta

Aplikací zaznamenané gesto je uloženo v souboru s příponou `.gst` pomocí XML serializace. Serializovaný objekt se zaznamenaným gestem je instance třídy `Gesture`. Příklad načtení zaznamenaného gesta uloženého do souboru ve vlastní aplikaci je uveden v ukázce 1.

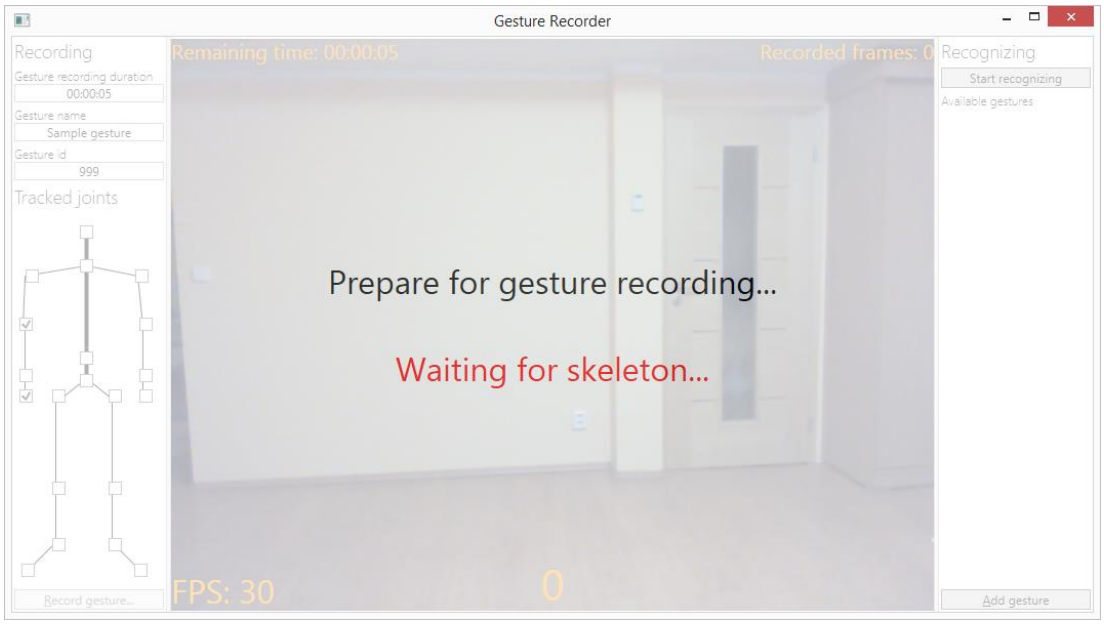
```
public Gesture LoadGestureFromFile(string fileName)
{
    XmlSerializer seralizer = new XmlSerializer(typeof(Gesture));
    Gesture gesture = null;

    using (Stream stream = File.Open(fileName, FileMode.Open))
    {
        gesture = seralizer.Deserialize(stream) as Gesture;
    }

    return gesture;
}
```

Ukázka 1: Načtení uloženého gesta.

Podrobnosti o objektu s uloženým gestem jsou k dispozici v referenční dokumentaci Mach.Kinect.chm uložené na přiloženém disku DVD.

2.5. Záznam gesta

Pro zaznamenání nového gesta je nutné nastavit časový úsek, po který bude zaznamenáváno, a vybrat body na postavě, které budou sledovány. Jméno a identifikátor gesta jsou volitelná pole a mohou obsahovat libovolnou hodnotu, která při vývoji vlastní aplikace usnadní identifikaci gesta po rozpoznání. Jako jméno může být libovolný test, jako identifikátor celé číslo.



Obrázek 2: Příprava před zaznamenáním nového gesta.

Kliknutím na tlačítko **Record gesture…** se ve střední části okna zobrazí odpočet, během kterého se uživatel provádějící gesto může připravit. Ihned po skončení odpočtu bude aplikace zaznamenávat pohyb zvolených bodů a ukládat jej po dobu nastavenou v políčku **Gesture recording duration**. V horním řádku je během nahrávání zobrazena informace o zbývajícím času záznamu gesta a počtu zaznamenaných pozic postavy.

Jakmile je nahrávání dokončeno, otevře se dialogové okno s možností uložení nahraného gesta. Po uložení se gesto rovnou umístí do seznamu v pravé části okna a je připraveno k rozpoznávání.

Na obrázku 2 je snímek obrazovky po zahájení nahrávání, ale žádný uživatel není senzorem sledován.

2.6. Rozpoznání gest

Pro rozpoznávání gest je nutné nejprve gesta přidat do aplikace. Automaticky jsou přidána gesta zaznamenaná během spuštěné instance editoru. Další gesta lze přidat pomocí tlačítka **Add gesture**.

Po přidání se v seznamu zobrazí jméno přidaného gesta, identifikátor a počet zaznamenaných rámců. Odebrat gesto z rozpoznávání lze přes kontextové menu zobrazené po pravém kliknutí na gesto v seznamu a vybráním možnosti **Delete**.

Samotné rozpoznávání je spuštěno kliknutím na tlačítko **Start recognizing…**.

Obrázek 3: Rozpoznané gesto zamávání pravou rukou.

Jméno posledního rozpoznaného gesta se zobrazí uprostřed náhledové obrazovky. Všechna rozpoznaná gesta jsou označena v levém sloupci šedým podkreslením. Na obrázku 3 je snímek obrazovky po rozpoznání jednoho gesta.