Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Bezkontaktní ovládání počítačových her

PLZEŇ, 2014 **PATRIK ŠOMA**

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V plzni dne 15. května 2014 ….……

vlastnoruční podpis

Poděkování

Děkuji Ing. Miroslavu Jiříkovi za vedení této bakalářské práce a za jeho připomínky během její vypracování. Také bych chtěl poděkovat za možnost zapůjčení Kinectu do vlastních rukou.

Abstrakt

Práce pojednává o různých způsobech bezdotykového ovládání počítačové hry. Tato hra je kompletně vyvinutá v jazyce Python za použití knihovny Pygame. Ovládání hry je realizováno pomocí senzoru Kinect, případně webové kamery. Při použití webové kamery je k detekci použita knihovna OpenCV. První část práce je teoretická a věnuje se knihovně Pygame, detekci obličeje pomocí OpenCV a technologii senzoru Microsoft Kinect. Druhá část se zabývá samotou realizací projektu. Jsou zde popsána pravidla a metody použité při vývoji hry.

Klíčová slova

Pygame, Kinect, OpenCV, detekce, bezkontaktní ovládání, haar features

Abstract

This thesis discusses various ways of touchless control over a computer game. This game is developed using programming language Python with Pygame libraries. Game control is implemented using Kinect sensor or webcam. When the webcam is in use program utilizes OpenCV library for detection. The first part of this thesis is theoretical and it is devoted to Pygame library, face detection with OpenCV and Microsoft Kinect sensor technology. The second part deals with realization of this project. All implemetation rules and methods used are described there.

Klíčová slova

Pygame, Kinect, OpenCV, detection, touchless interface, haar features

Obsah

1 Úvod

Umělá inteligence jako vědní obor je velmi obsáhlá disciplína, která nemá jasně vytýčené hranice. Definice „klasické“ lidské inteligence je stále předmětem neustálého zkoumání, a proto je velmi složité vytvořit jasný popis. Několik vědců se pokusilo inteligenci definovat, ale je těžké najít dva stejné názory. V jednom se nicméně shodli, inteligence je schopnost řešit neznámé problémy a učit se ze zkušeností. Snaha vytvořit něco takového uměle je velice lákavá a člověka vždy fascinovala. Důkazem toho je samotná historie, kde lze sledovat různé pokusy o napodobení lidské bytosti, ať už jde o fikci nebo skutečnost, ne jenom jako stroj vykonávající opakující se práci, ale jako samostatně myslící výtvor.

V roce 1950 popsal Alan Turing v článku „Computing Machinery and Intelligence“ test umělé inteligence, který vypadá přibližně takto: Existují dvě oddělené místnosti. V první sedí člověk a ve druhé robot disponující umělou inteligencí. Mimo tyto místnosti je zkoušející, který dovnitř posílá otázky, na které očekává odpovědi. Pokud zkoušející z odpovědí nerozpozná robota od člověka, přičte si robot bod inteligence. Zjednodušeně se dá říct, že pokud lingvistika robota napodobuje lingvistiku člověka, lze prohlásit robota inteligentním. Proti tomuto testu byl často pokládán argument čínského pokoje, který říká, že pokud má umělá inteligence k dispozici všechna data k řešení různých problémů, ale zároveň nechápe jejich význam tak se jedná pouze o prohledávání stavového prostoru a ne o skutečně silnou umělou inteligenci. Vyvstává tedy otázka, zdali chceme pouze napodobovat uvažování člověka nebo vytvořit zcela samostatně myslící entitu.

Zatím je velmi nepravděpodobné, že se podaří vytvořit umělou inteligenci, která bude absolutně obecná a dokáže přijmout libovolnou formu vstupu, ke kterému vytvoří smysluplný výstup. I přes toto omezení je možné zkoumat určité oblasti umělé inteligence, které se zabývají jen jistým typem řešení. Automatické zpracování obrazu, tvorba neuronových sítí a expertní systémy jsou jen příklady úloh, kde se setkáváme s jistým úspěchem v tvorbě umělé inteligence.

Cílem této práce je náhlednou do bezkontaktní komunikace s počítačem a seznámit se s nástroji pro vizualizaci a zpracování obrazu. Získané znalosti pak aplikovat při vytvoření hry. Tato hra nabízí možnost výběru mezi ovládáním obličejem a ovládáním rukou za použití různých technologií a detekčních metod. Aplikace je klonem klasické hry Arkanoid.

2 Teoretická část

Tato kapitola se věnuje teorii potřebné k vytvoření hratelné aplikace. Následující odstavce jsou nejen o historii každé důležité části použité při vývoji, ale také nahlížejí do funkčnosti jednotlivých metod detekce.

2.1 Pygame

Pygame je balíček modulů navrhnutý pro psaní počítačových her v jazyku Python. Tyto moduly přidávají funkce, které usnadňují vytvoření hry a dalších multimediálních programů. Je použitelný ve většině operačních systémů, a proto je ideální pro projekty, na kterých se účastní více vývojářů. Následující podkapitoly se věnují nejdůležitějším použitým modulům.

2.1.1 Surface

Surface je modul, který na sobě umožňuje zobrazovat barvy a obrázky. Je to základní kámen pro práci v Pygame. Surface (povrch) je plocha, která může být jakkoliv velká a lze jich vytvořit libovolný počet. Používá se jako objekt pro zobrazení předmětů ve hře. Lze ho vytvořit následujícím příkazem:

$$\boldsymbol{povrch} = \boldsymbol{pygame.Surface((width, height))} \qquad (2.1)$$

2.1.2 Display

Speciální případem surface je objekt display, který reprezentuje obrazovku a vymezuje herní prostor pro všechny ostatní surface. Do tohoto okna se vykreslují všechny herní objekty, statické i dynamické. Display musí v každém časovém okamžiku vykreslit všechny objekty, jinak nebudou ve hře vidět. Pygame si nedokáže objekt prostě zapamatovat a "držet" objekt na pozici dokud není vyžadovaná jeho další akce. Proto každá takto naprogramovaná hra, musí pracovat v nekonečném cyklu zajišťující neustálé obnovování obrazu. Tato komponenta také udává souřadnicový systém pro celou aplikaci. Velikost okna je volitelná, důležitý je levý horní roh, zde jsou souřadnice X a Y nulové, směrem

dolů nabývá souřadnice Y kladných hodnot a směrem do prava zase nabývá souřadnice X kladných hodnot. Bez tohoto modulu nelze hru vytvořit. Inicializace se provádí následovně:

$$\boldsymbol{obrazovka} = \boldsymbol{pygame.display.set\_mode}((\boldsymbol{width}, \boldsymbol{height})) \quad (2.2)$$

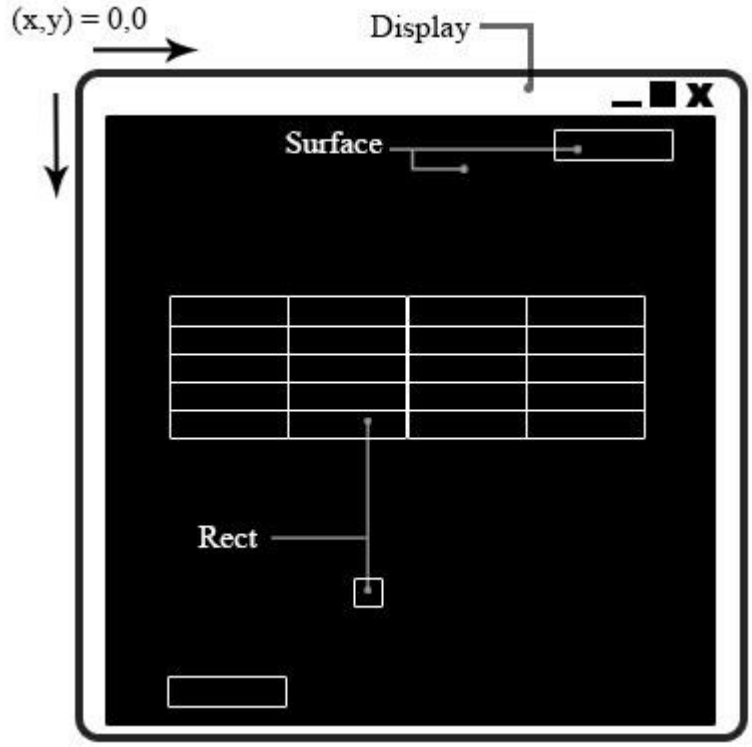
Pygame může v jednu chvíli zobrazovat pouze jeden Display, vytvořením dalšího způsobíme zavření toho původního. Modul si také hlídá takzvané „events“ neboli události. Okno tak dokáže reagovat na vstupy, kterými mohou být například kliknutí myši nebo stisk klávesnice. Display je také zodpovědný za udržení správného rozlišení v případě změny velikosti okna, například na celou obrazovku.

2.1.3 Rect

Rect je modul, který umožňuje manipulaci s objekty ve tvaru obdélníku. V chování je podobný modulu Surface, také se jedná o obdélníkovou plochu, na kterou lze zobrazit obrázky a barvy. Nicméně surface je obyčejná „nehmotná“ plocha navrhnutá hlavně pro zobrazování grafiky, nikoliv pro interakci s hráčem. Rect nabízí několik implementovaných metod, které ulehčují práci se segmenty hry, které vyžadují náročnější výpočty. Příkladem může být velice často řešený problém kolize dvou předmětů. Matematické řešení kolize mnoha objektů ve hře může nepříjemně ovlivnit plynulost aplikace, Rect dokáže kolizi detekovat velice efektivně a bez zbytečných časových prodlev. Kromě metod detekce kolize, dokáže tento modul s předměty pohybovat, plynule měnit jejich velikost atd. Vytvoření objektu se provádí následovně:

$$\boldsymbol{objekt} = \boldsymbol{pygame.Rect}(\boldsymbol{left}, \boldsymbol{top}, \boldsymbol{width}, \boldsymbol{height}) \quad (2.3)$$

Lze vidět, že k vytvoření nám stačí znát počáteční polohu levého horního rohu objektu a jeho šířku a výšku. Na rozdíl od Surface, kde počáteční polohu můžeme udat až při vykreslování povrchu.



Obrázek 2.1 – Rozložení objektů v aplikaci

2.2 OpenCV

OpenCV neboli Open Source Computer Vision je volně dostupná knihovna mířená hlavně na práci s počítačovým viděním a zpracováním obrazu v reálném čase. Knihovna je šířena pod BSD licencí, a tak je možné ji využít ke studijním i komerčním účelům. OpenCV má C++, C, Python, MATLAB a Java rozhraní a podporuje systémy Windows, Linux, iOS, Mac OS a Android. Obsahuje přes 2500 algoritmů optimalizovaných pro různé druhy strojového vidění a učení. Umožňuje rozeznávání a detekci obličejů, sledování pohybu, extrakci 3D objektů a podobně.

Knihovna je napsaná v optimalizovaném C/C++ a dokáže využít více jádrové procesory. Díky tomu je OpenCV vyhledávané ve velkých firmách (Google,

Yahoo, Microsoft, Intel,…), které ji využívají při vývoji vlastních aplikací. V této práci se OpenCV využívá pro jeho vlastnost detekovat obličej.

2.2.1 Historie

Původně vyvíjeno společností Intel v roce 1999 jako vylepšení aplikací využívající CPU. Tyto aplikace byly součástí projektu zabývající se metodou ray-tracking při konstrukci obrazu. Vývoj se orientoval na zkoumání vidění a vytvoření optimalizovaného kódu, který by poskytnul základní infrastrukturu v oblasti vidění. Cílem bylo rozšíření této infrastruktury a poskytnou tak vývojářům společný základ, na kterém by mohli postavit další vývoj. Aplikace, které by takto vznikly, by byly přenosné, multi platformní a zdarma.

První verze OpenCV byla představena v roce 2000 na konferenci o strojovém vidění a rozeznávání vzorů na IEEE (Institute of Electrical and Electronics Engineers). Do roku 2005 bylo dostupných pět beta verzí OpenCV, v roce 2006 vznikla první plnohodnotná verze. V roce 2008 se OpenCV dočkalo podpory od společnosti Willow Garage, která ještě v tom samém roce umožnila vydání verze 1.1 .

V dalším roce vznikla verze 2.0, která se zaměřila hlavně na optimalizaci knihovny a vylepšení výkonů na více jádrových systémech.

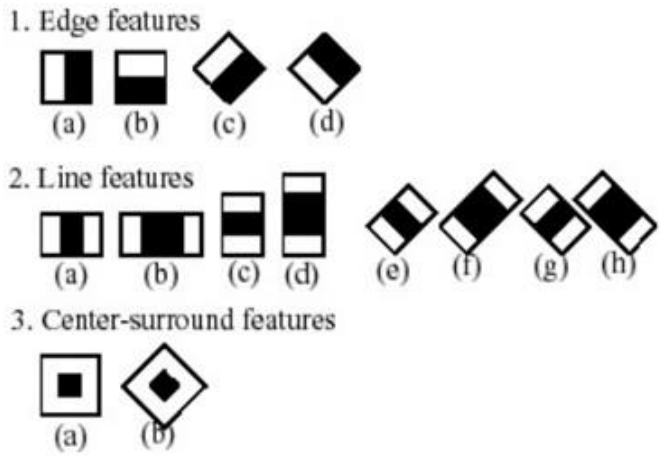
V roce 2012 bylo OpenCV převzato neziskovou společností Opencv.org, která poskytuje podporu do dnešního dne.

2.2.2 Detekce obličeje

Pro detekci obličeje využívá OpenCV kaskádový klasifikátor natrénovaný pomocí takzvaných Haar features. Jedná se o velmi efektivní metodu též známou jako "Viola - Jones object detection framework". Hlavní myšlenkou je vytvořit slabé klasifikátory, které jsou natrénovány pomocí pozitivních a negativních obrázků. Následně pak jejich lineární kombinací vytvořit silný a výpočetně nenáročný klasifikátor schopný velmi přesné detekce v reálném čase. Metoda musí překonat několik překážet, které jsou probrané dále.

2.2.3 Haar features

Prohledávání obrazu a vyhledání požadovaného objektu (tvaru) se provádí pomocí Haar features. Název Haar odkazuje na matematické funkce čtvercového tvaru a features odkazuje na rysy. Jedná se o jakousi bodovou mapu, ve které každé pole má hodnotu kladnou (+1) nebo hodnotu zápornou (-1). Tvary map se liší podle druhu detekce, například pro jednoduché hledání okrajů v obraze budou použité mapy jednodušší a bude jich menší počet.



Obrázek 2.2 – Typy Haar features [3]

Pokud chceme detekovat část obličeje (oči, nos,…) používají se features pro extrakci hodnot z obrazu, které se potom porovnávají s natrénovanou množinou. Haar features mají různé tvary, které využívají typické rysy pozorované pro požadovanou detekci. Pokud bych chtěl například detekovat nos, využiji toho, že nos je často světlejší než okolní tváře.

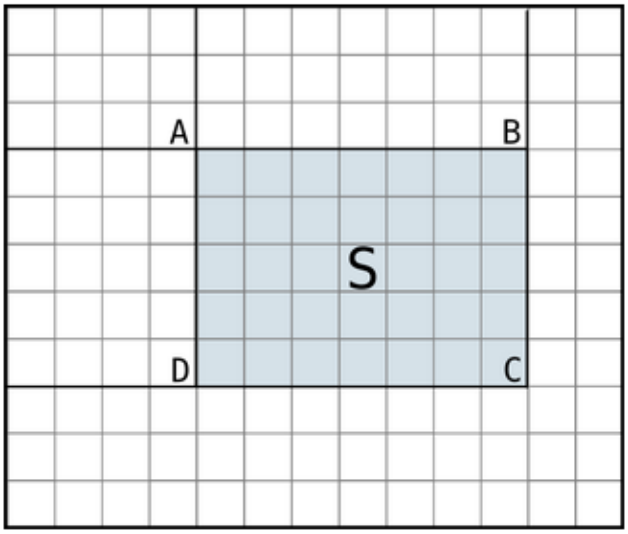
Obrázek 2.3 – Aplikace Haar features při detekci [3]

Zvolené Haar features potom zkoušíme přikládat na obraz ve všech možných pozicích a velikostech. Po přiložení se sleduje, zda oblast odpovídá použité feature a pokud ano dojde k sečtení všech pixelů pod černou (kladnou) oblastí a bílou (zápornou) oblastí. Získáme, tak jednu hodnotu, která bude kladná, nebo záporná. Rozložení a velikost těchto hodnot potom buď odpovídá natrénování, nebo ne.

Tuto metodu detekce, nicméně doprovází několik problémů. Tím hlavním je náročnost, pokud se Haar features přikládají v různých velikostech a pozicích, musí se neustále sčítat velké množství pixelů. Z tohoto důvodu byly to metody zavedeny *integral images*.

2.2.4 Integral images

Výpočetní náročnost detekce lze snížit pomocí integral images neboli integrálních obrazů. Jelikož Haar features mají obdélníkový tvar, nabízí se možnost nesčítat všechny body v detekované oblasti, ale využít pouze "rohy". K sestrojení obdélníku vlastně stačí pouze čtyři body, a protože oblast obsahuje pouze hodnoty +1 nebo -1 není třeba neustále procházet každý pixel zvlášť, ale pouze jednou vytvořit integrální obraz.



Obrázek 2.4 – Princip výpočtu integrálního obrazu [6]

$$\boldsymbol{S = A - B - C + D}$$

Rovnice 2.1 – Obsah integrálního obrazu

Integrální obraz vznikne přepočítáním vstupního obrazu. Každá pozice integrálního obrazu se získá ze stejné pozice vstupního obrazu a to tak, že sečteme všechny pixely na stejné řádce až do požadovaného bodu, to samé platí pro sloupce vstupního obrazu. Po vynásobení těchto dvou čísel je získána požadovaná hodnota.

$$I(x,y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y')$$

Rovnice 2.2 – Výpočet pro vytvoření integrálního obrazu

Original image

1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

Integral image of original image

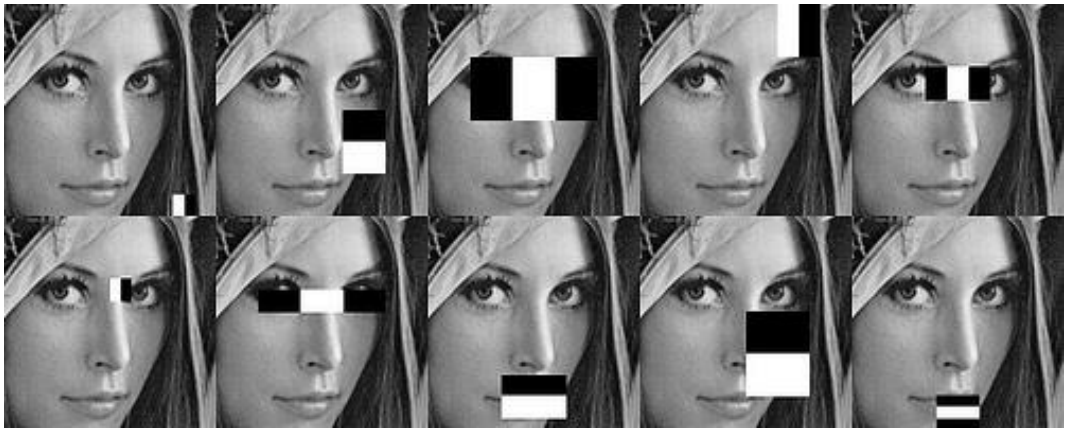
1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	14	16	18	20	22	24
3	6	9	12	15	18	21	24	27	30	33	36
4	8	12	16	20	24	28	32	36	40	44	48
5	10	15	20	25	30	35	40	45	50	55	60
6	12	18	24	30	36	42	48	54	60	66	72
7	14	21	28	35	42	49	56	63	70	77	84
8	16	24	32	40	48	56	64	72	80	88	96
9	18	27	36	45	54	63	72	81	90	99	108
10	20	30	40	50	60	70	80	90	100	110	120
11	22	33	44	55	66	77	88	99	110	121	132
12	24	36	48	60	72	84	96	108	120	132	144

Obrázek 2.5 – Příklad integrálního obrazu [10]

Na okrajích oblasti jsou takto vlastně již pixely spočítané a po sečtení "rohů" získáme výslednou hodnotu.

2.2.5 Adaboost

Kromě výpočetní náročnosti má tato detekce problém s redundancí haar features. Při každém přiložení se zkouší, zda k detekci došlo nebo ne. Je logické, že pokud na obrázku hledáme pár očí, nedojde na většině pozic k detekci. Pokud se využívá velké množství různých typů Haar features, některé jsou pro určité detekce užitečnější než jiné, a proto budou mít vybrané typy (tvary) větší úspěšnost než jiné. Ty méně potřebné lze vyřadit.



Obrázek 2.6 – Putování Haar features na vstupním obrazu [7]

Adaboost je algoritmus, který se pokouší naučit které features se při detekci hodí více. Při trénování se odstraní takové features, které měli příliš velkou chybovost pro konkrétní detekci. Po nalezení těch nejlepších vznikne takzvaný *slabý klasifikátor*. Ten dokáže klasifikovat pouze o něco lépe než náhodné rozhodování, zpravidla platí, že klasifikátor musí správně zaklasifikovat více než polovinu předložených vstupů aby mohl být požadován alespoň za slabý.

Pokud existuje množina těchto slabých klasifikátoru, lze jejich kombinací získat klasifikátor silný.

Při trénování vezme Adaboost klasifikátor, který měl při výběru features nejmenší chybovost. Trénovací obrazy získají váhové ohodnocení, které je uniformně rozděleno. Klasifikátor se vyzkouší na trénovací množině a sleduje chybně zaklasifikované obrazy. Tyto obrazy získají větší váhu než ostatní, potom se postup opakuje. Pokaždé se podaří množinu obrazů rozdělit jednou lineární funkcí, nicméně lineární kombinací všech předchozích pokusů lze docílit *silného klasifikátoru*, který by měl být schopný správně klasifikovat většinu obrazů.

Kombinací těchto vlastností dokáže OpenCV detekovat v reálném čase několik prvků najednou.

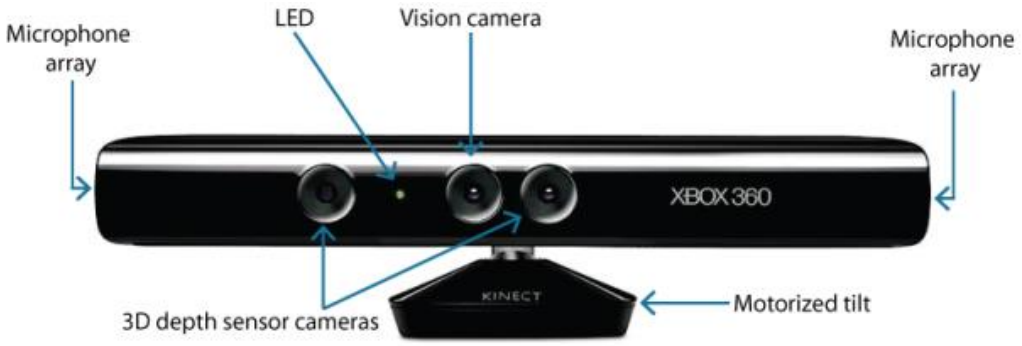
2.3 Microsoft Kinect

Kinect byl vyvinut společností Microsoft pro herní konzoli Xbox 360 jako revoluční nástroj pro bezdotykové hraní her. Projekt byl původně známý pod krycím jménem "project Natal" a byl spuštěn jako odpověď na předvedení herní konzole Wii, kterou vyvinula v roce 2005 společnost Nintendo.

Nintendo Wii, které obsahuje ovládání s názvem Wii Remote dokáže detekovat pohyb ve třech osách. K tomu mu pomáhá optický senzor, který rozpozná, kam ovladač zrovna míří.

Microsoft se chtěl se svým produktem posunout ještě trochu dále a oprostit uživatele od všech fyzických předmětů (ovladačů). Uživatel by tak získal absolutní volnost pohybu a zároveň docílil maximálního pohodlí.

Kinect je senzor, který se skládá ze tří podsystémů. Prvním je klasické počítačové vidění neboli RGB kamera, která je prakticky stejná jako klasické webkamery integrované v dnešních počítačích. Druhým podsystémem je hloubkové vidění, které je kriticky důležitou vlastností jak pro bezdotykové hraní her, tak i umělou inteligenci obecně. Posledním podsystémem je rozpoznávání zvuku a řeči. Kinect má tedy sloužit jako úplný balíček funkcí, které jsou potřeba k bezdotykovému ovládání aplikací.



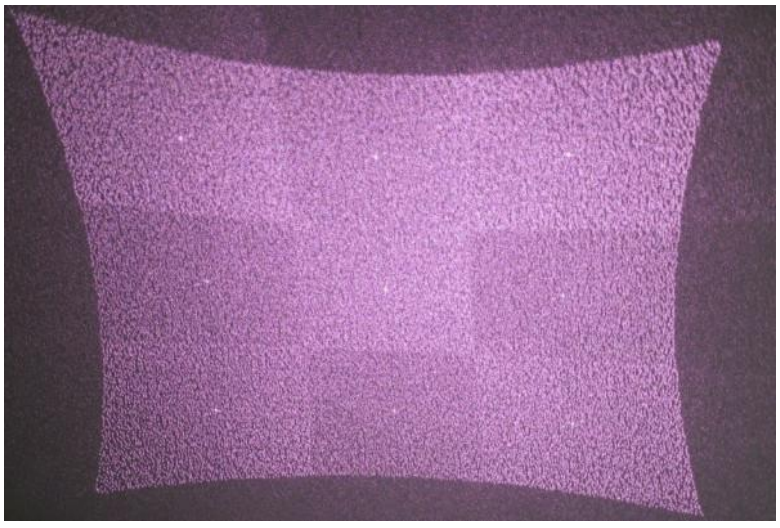
Obrázek 2.7 – Jednotlivé části Kinectu [9]

Toto všechno bylo nejdříve zamýšleno pouze pro Xbox 360, ale s narůstající komunitou vývojářů si společnost Microsoft uvědomila, že využití takového senzoru i mimo herní prostředí je ohromná příležitost. Není tedy překvapením, že po nějaké době byl k dispozici i Kinect pro operační systém Windows s vlastním SDK.

2.3.1 Hloubkový senzor

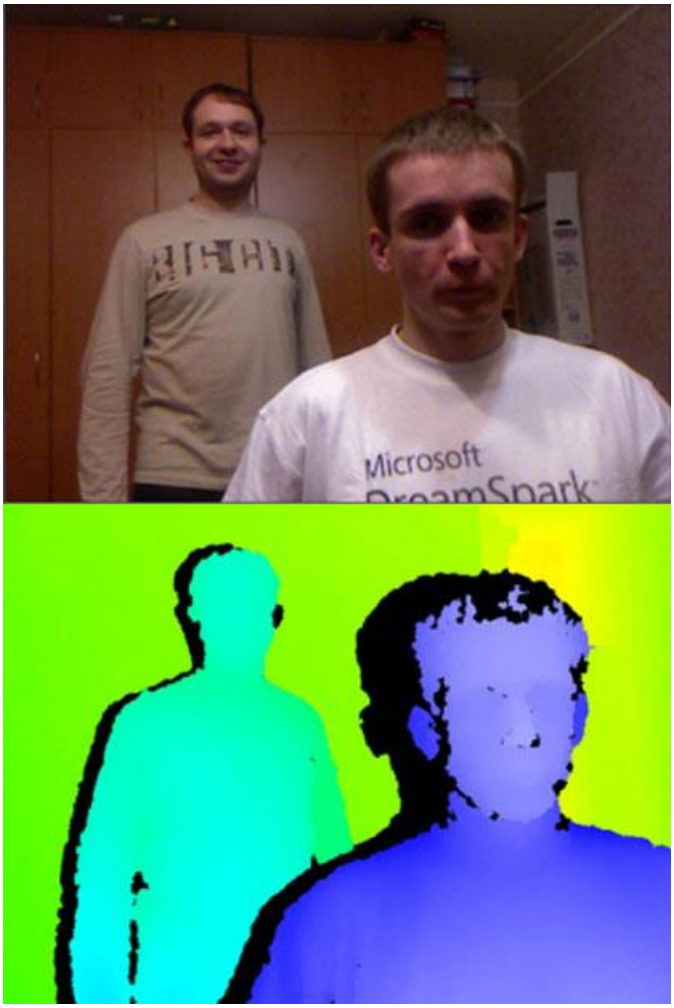
Hloubkový senzor se skládá ze dvou částí. První je vysílač infračerveného (IR) záření a druhou je IR přijímač. Infračervené záření je pro lidské oko neviditelné a zároveň přístrojově lehce detekovatelné.

Vysílač promítne síť několika stovek teček. Nejedná se o symetrickou síť, kde by všechny tečky měly od sebe stejnou vzdálenost. Tečky jsou rozděleny do skupin, tyto skupiny mají každá unikátní rozdělení teček a jejich vzdáleností od sebe. Kinect má apriorní znalost o tom jak každý shluk vypadá a kde je jeho pozice v síti. Po promítnutí do prostoru se tečky budou odrážet od předmětů před Kinectem. Infračervený senzor potom hledá známé shluky teček a kontroluje, pod jakým úhlem na něj dopadají. Známý je tedy úhel vysílání a úhel přijímání, nyní lze zjistit vzdálenost předmětu od senzoru. Výsledný obraz, který vznikne po zmapování scény se nazývá hloubková mapa.



Obrázek 2.8 – Síť IR bodů [11]

Hloubková mapa má rozlišení 640x480 pixelů a snímání probíhá v režimu 30 FPS (snímky za vteřinu). Hardware je schopný poskytnou obraz o rozlišení až 1280x1024, ale za cenu nižšího FPS.



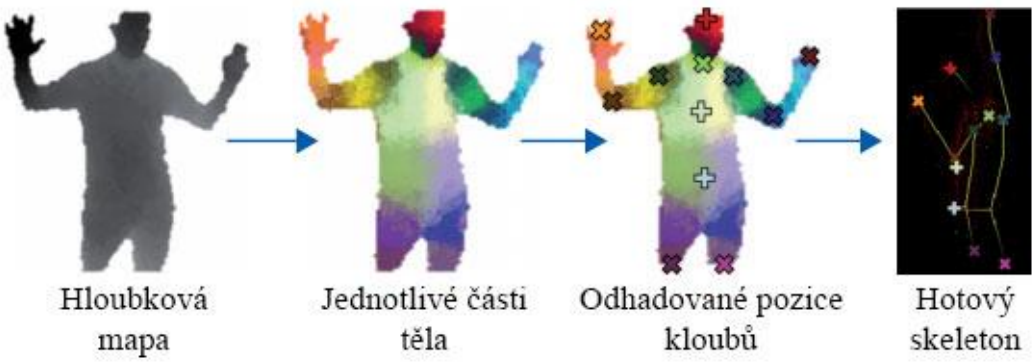
Obrázek 2.9 – Převod vstupního obrazu na hloubkovou mapu [12]

Hloubkový senzor v Kinectu pro Windows pracuje ve dvou módech, prvním je default mode a druhým je near mode. Default mode pracuje ve vzdálenosti 0,8 m do 4 m, near mode funguje v rozmezí 0,4 m až 3 m od senzoru. Tyto módy se uplatní hlavně při detekci a sledování lidské postavy.

2.3.2 Skeletal tracking

Reprezentace lidského těla je realizována pomocí jeho skeletonu. Hloubková mapa poskytuje pouze náhled na podobu scény, nicméně neposkytuje informaci o obsahu uvnitř. Kinect je natrénován tak, aby dokázal rozeznat různé druhy lidských postav v odlišných pózách. Dosaženo je toho díky obsáhlému natrénování klasifikátoru, který dokáže správně klasifikovat nejčastější pózy člověka při hraní her. Použitý klasifikátor je také částečně schopný rozpoznat pózy, pro které nebyl natrénován a přizpůsobit se tak různým úhlům uložení senzoru před uživatelem. Ideálně by si Kinect neměl všímat nelidských pohybujících se objektů (dekorace, domácí zvířata, …).

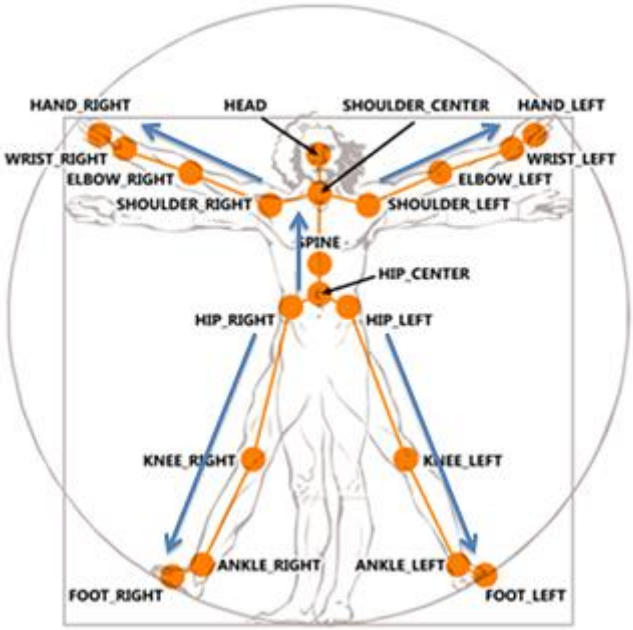
Při detekci dokáže Kinect rozdělit lidské tělo na části ve kterých lze očekávat kloub a na části, které pouze vyplňují místo mezi klouby. Výstupem pak jsou prostorové souřadnice dvaceti lidských kloubů, které dohromady tvoří skeleton reprezentující celé lidské tělo.



Obrázek 2.10 – Rekonstrukce skeletonu [1]

Algoritmus pro sledování skeletonu je navrhnutý tak, aby dokázal sledovat až šest lidí zároveň a ze dvou generoval skeleton. Sledování probíhá pouze tehdy, pokud uživatel směřuje celým tělem přímo ke Kinectu. Existuje jistá tolerance natočení, ale sledování těla z profilu není možné kvůli absenci lidských částí v obraze.

Pokud Kinect obsahuje default a near mode, probíhá sledování podle zvoleného módu. V default módu hledá Kinect všech dvacet kloubů, near mode sleduje klouby od pasu nahoru.



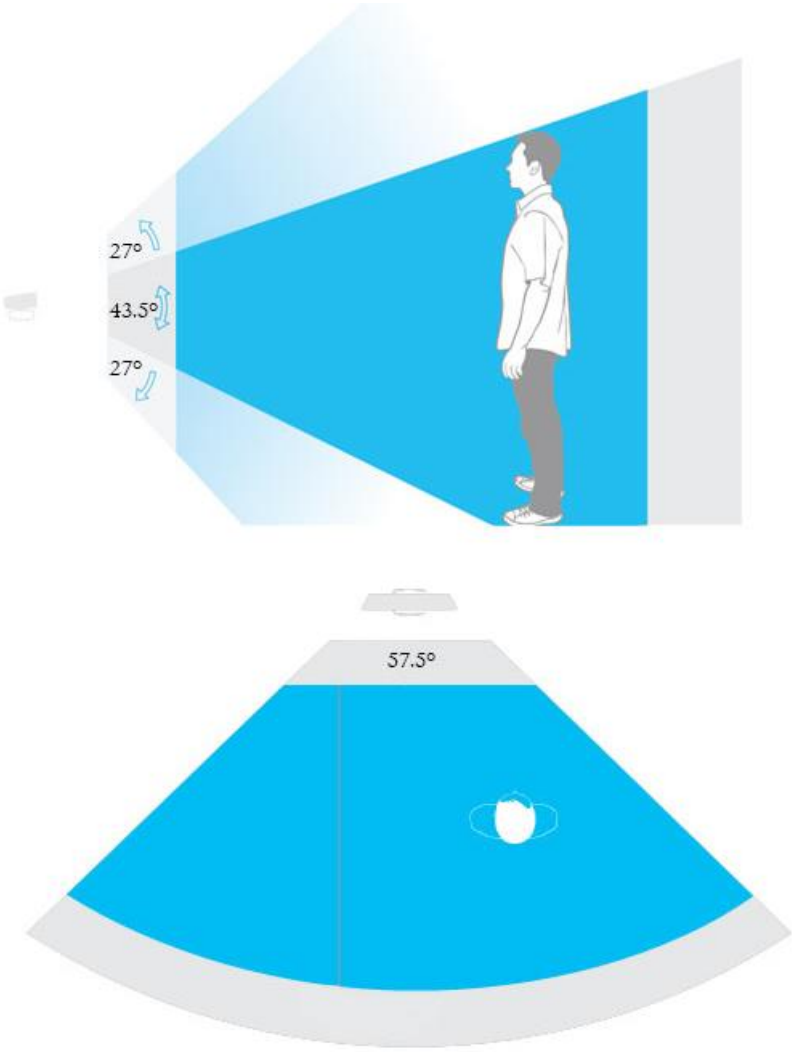
Obrázek 2.11 – Klouby skeletonu [7]

Tento postup detekce zaručuje určité výhody oproti detektorům, které nemají „smysl“ pro hloubku v prostoru. Největší je nezávislost na okolním osvětlení, Kinect funguje i v absolutní tmě. Nicméně intenzivní sluneční záření naopak dokáže „přesvítit“ IR záření z kinectu. Lze aplikacím nastavit, jaké maximální vzdálenosti si mají všímat.

2.3.3 Zorné pole

I když dokáže Kinect provádět detekci ze široké škály úhlů a vzdáleností, podléhá jistým omezením. Hloubkový senzor a RGB kamera spolu hodně spolupracují, a proto jsou pro ně snímací zorná pole stejná. V horizontálním pohledu funguje Kinect v rozsahu 57,5 stupňů, ve vertikálním je to 43,5 stupňů.

Zorné vertikální pole lze posunout směrem nahoru nebo dolů o 27 stupňů pomocí zabudovaného motoru, který při hraní automaticky pomáhá najít ideální úhel mezi podlahou a uživatelem tak, aby detekce byla co nejpřesnější. Tato omezení musí samozřejmě respektovat i limity hloubkového senzoru (2.3.2).



Obrázek 2.12 – Zorné pole senzoru Kinect [7]

3 Praktická část

Tato část je věnována samotné realizaci hry a popisu metod použitých při vývoji. Cílem je vytvořit jednoduchou hru, kterou dokáže každý pochopit a zároveň bude mít intuitivní ovládání. Z těchto důvodů byl zvolen klon klasické hry Arkanoid, který je na veřejnosti dobře známý a nevyžaduje speciální znalosti ani dovednosti. V rámci projektu by mělo být možné hru později jednoduše rozšířit pro více hráčů. Jelikož je hra napsaná v jazyce Python za použití knihovny Pygame, byla pojmenována Pycanoid.

3.1 Návrh

Vzhledem k tomu, že nápad na hru vzniknul díky návrhu na projekt spojený s vyhlášením Plzně jako hlavního města kultury pro rok 2015, musí dodržovat určitá omezení. Navrhnutým projektem je komunikacemi mezi dvěma městy pomocí projektoru přímo na ulici. Jako doplněk může sloužit právě Pycanoid, který by mohl být ovládán z obou stran „komunikačního okna“.

Nejdůležitější je hru vytvořit tak, aby mohla fungovat bez dozoru a zároveň nemusely být připojeny kromě Kinectu žádné další periferie (klávesnice, myš). Uživatel by měl být schopný přijít, odehrát hru a dát prostor dalšímu. Pycanoid sice nabízí i ovládání pomocí myši, ale hlavním prostředkem je Kinect. Absence klávesnice znemožňuje jakékoliv zadávání údajů, a proto musí být automatická celá logika hry, nejenom její ovládání.

Pycanoid může fungovat v nepřetržitém provozu, pokud by se používal na veřejném místě, musí se pomocí kalibrace, popsané dále, vymezit herní prostor tak, aby přihlížející lidé nemohli ovlivnit dění ve hře.

Při samotném hraní uživatel ovládá najednou dvě desky pro odrážení „kuličky“, jedna ve spodní části obrazovky a druhá v horní části. Klasický Arkanoid tuto možnost nenabízí, zde byla deska navíc přidána z důvodu případného hraní dvou hráčů zároveň. Pokud by tedy opravdu došlo k tomu, že bude Pycanoid použit při meziměstské komunikaci, lze ho rychle přizpůsobit.

Cyklus hry je jednoduchý, uživatel si stoupne před Kinect a pomocí pravé ruky může ihned pohybovat odrážecími deskami. Pohybem vzhůru odpálí míček a pokusí se co nejrychleji porazit všechny překážky. Až se tak stane, hra vypočítá skóre a uloží všechny nasbírané informace pod unikátním jménem. Jméno se generuje jako unikátní číslo odvozené od aktuálního času.

3.2 Generování překážek

Jelikož je hra navržena pro neustálý provoz, provádí se generování překážek náhodně. Prvním krokem je vytvoření mapy překážek. Na začátku každé hry se vygeneruje matice, která se náhodně vyplní čísly 0 nebo 1. Číslo 1 reprezentuje překážku a 0 reprezentuje prázdné místo. Matice se ukládá do externího souboru. Při generování grafiky na obrazovku se matice načte ze souboru. Program pak podle počtu řádek a sloupců matice spočítá velikost překážkových bloků tak, aby se přesně vešly do vytýčené oblasti.

Ukládání do externího souboru je pro případ budoucí úpravy hry. V případě, že by bylo potřeba rychle pozměnit logiku hry tak, aby bylo možné předdefinovat různé úrovně.

Následuje algoritmus, který postupně načítá mapu a podle spočítaných souřadnic nakreslí do hry blok nebo vynechá prázdné místo.

3.3 Odrazy

Pycanoid je hra plná odrazů i když se na pohled zdá, že se jedná o jednoduchý odraz typu „úhel dopadu rovná úhlu odrazu“, není tomu vždy tak. Desky ovládané uživatelem, disponují takzvaným „kulatým odrazem“. Uživatel musí mít možnost během hry měnit dráhu míčku tak, aby mohl efektivněji ničit překážky. Absence kulatého odrazu by způsobila, že by míček uvízl v opakujících se odrazech. Navíc výsledné skóre by bylo prakticky závislé na náhodně vygenerovaném úhlu při prvním odpalu.

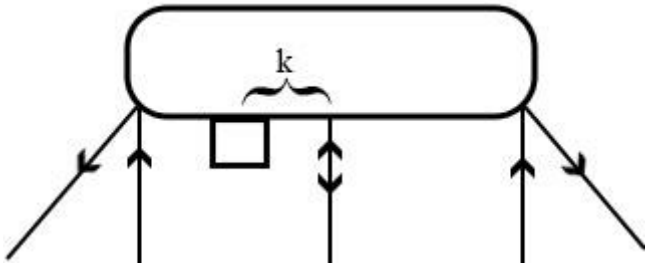
„Klasický“ odraz je aplikován při kolizi kuličky se stěnou nebo s překážkovým blokem. Hra si uchovává informaci o aktuálním úhlu, ve kterém se kulička

momentálně nachází a pomocí něho vypočítává nový úhel. Výpočet neobsahuje goniometrické funkce a úhel může nabývat záporných hodnot, pokud tomu odpovídá směr pohybu kuličky.

$$\boldsymbol{a)\ \varphi_{nový} = \varphi - (2 * \varphi)} \qquad \boldsymbol{b)\ \varphi_{nový} = (90 - \varphi) + 90}$$

Rovnice 3.1, 3.2 – a) výpočet pro horizontální odraz b) výpočet pro vertikální odraz

Kulatý odraz využívá podobné rovnice pro výpočet nového úhlu při horizontálním odrazu, nicméně byla přidána konstanta, která určuje, jak moc se úhel změní. V případě dopadu kuličky na desku se měří vzdálenost středu desky od středu kuličky. Při tomto odrazu jsou důležité hlavně okraje desky, protože tam by mělo docházet ke změně úhlu a směrem ke středu by měl úhel odrazu stále více odpovídat úhlu dopadu.



Obrázek 3.1 – Kulatý odraz

$$\boldsymbol{d} = \frac{\left(\frac{\boldsymbol{l}}{\boldsymbol{2}}\right) * \boldsymbol{k}}{\boldsymbol{100}} \quad kde\ l = délka\ desky, k = vzdálenost\ od\ středu$$

Rovnice 3.3 – Konstanta pro výpočet změny úhlu odrazu

Ze vzorečku je vidět, že pokud je *k* nulové, došlo dotyku ve středu, tak je celá konstanta *d* nulová. Jakmile se začne vzdálenost zvětšovat, dostaneme i větší *d.*

$$\boldsymbol{\varphi_{nový} = \varphi - ((2 + d/2) * \varphi)}$$

Rovnice 3.4 – Rovnice pro kulatý odraz

Zavedením *d* do rovnice odrazu je docíleno kulatého odrazu, který napodobuje zaoblený tvar hrací desky.

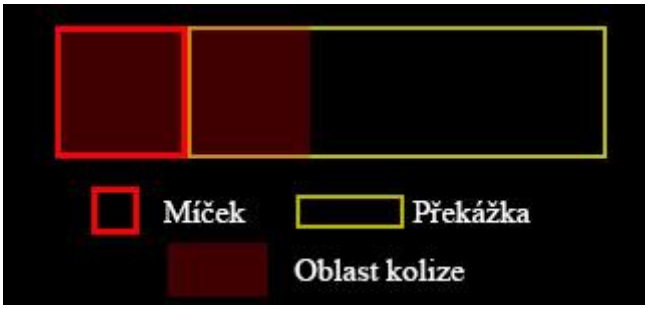
3.4 Detekce kolize

Při kolizi se nabízí možnost kontrolovat každý bod na okrajích kuličky oproti okrajům všechny překážek, nicméně tato varianta je příliš náročná a zpomaluje chod programu. Také by bylo nutné si neustále udržovat souřadnice všech bloků a po každém nárazu je aktualizovat.

Pygame nabízí možnost automatické detekce kolize. Překážky a míček jsou objekty typu Rect. Při generování grafiky se z mapy překážek stávají grafické překážky (viz. 3.2). V tom samém okamžiku se vykreslované objekty přidávají do seznamu. Za pomocí funkce *collidelist* lze zjistit, zda jeden Rect objekt je v kolizi s jiným objektem uloženým v seznamu. Funkce vrací pořadí objektu v seznamu. Informace o zasaženém bloku spolu s jeho souřadnicemi se načte do logické části programu.

Následně se kontrolují středy všech stran míčku a oproti zasažené oblasti překážky. Díky tomu, že se kontrolují všechny hrany zvlášť lze říct, z jaké strany náraz přišel a aktivovat k tomu patřičný odraz. Protože ale není možná bodová detekce kolize, musí se vytýčit jisté toleranční pásmo, ve kterém lze říct, že kulička pravděpodobně již narazila. Míček se sice pokaždé neodrazí přesně od stěny bloku, ale tato nepřesnost je lidským okem nepozorovatelná.

Tato metoda je mnohem méně náročná než kontrola jednotlivých pixelů. Celá aplikace tak pracuje plynule nehledě na počet překážek.



Obrázek 3.2 – Příklad oblasti možné kolize

Po odrazu se detekovaný blok vyhledá v mapě překážek. Odpovídající hodnota se změní z 1 na 0, tak při vykreslování dalšího snímku hry bude upravená mapa odpovídat zasažení. Po poražení všech překážek se vygeneruje kompletně nová mapa a hra se opakuje.

3.5 Kalibrace souřadnic

Kalibrace je důležitá z důvodu správného převedení jednoho souřadnicového systému na jiný. Jelikož OpenCV, Kinect a Pygame využívají různé souřadnicové systémy, nelze bez kalibrace ihned ovládat hru snímanou částí těla.

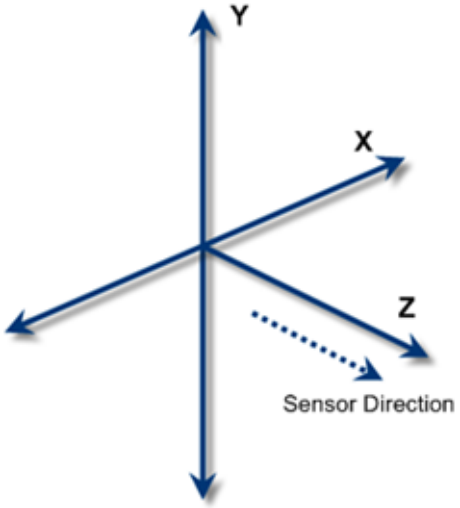
Ideálně by se kalibrace měla provádět před každou hrou, aby nastavení bylo individuální, nicméně vzhledem k tomu, že hra bude mít fyzicky vytýčený prostor pro hraní, bude stačit kalibrovat pouze podle potřeby.

OpenCV má stejně orientovaný souřadný systém jako Pygame to znamená, že počátek systému je v levém horním rohu a směrem na pravou stranu se zvětšuje souřadnice x a směrem dolů se zvětšuje souřadnice y. Nicméně rozlišení, které poskytuje webkamera může být různé a nikdy nebude přesně stejné jako má Pycanoid. Okno Pycanoidu má výchozí rozlišení 1024x768 pixelů, ale hrací plocha nezabírá celé okno. Hra je také navržena tak, aby rozměry byly co nejvíce dynamické a při změně rozlišení se automaticky změnili velikosti všech objektů ve hře. Proto není možné nikdy přesně říct, jak velká bude hrací plocha.

Pokud pomocí OpenCV zobrazíme obraz webkamery zjistíme, že není zrcadlový. Nejjednodušší způsob jak synchronizovat pohyb obličeje a herní desky je tedy

obraz otočit. Pygame takovou možnost nabízí, ale otočil se tím i celý souřadnicový systém.

Kinect využívá vlastní, kompletně odlišný systém. Díky prostorovému vnímání pracuje senzor s třemi souřadnicemi. Počátek souřadnic je uprostřed obrazu, který Kinect snímá. Pohyb na pravou stranu inkrementuje souřadnici x, pohyb vzhůru zase souřadnici y.



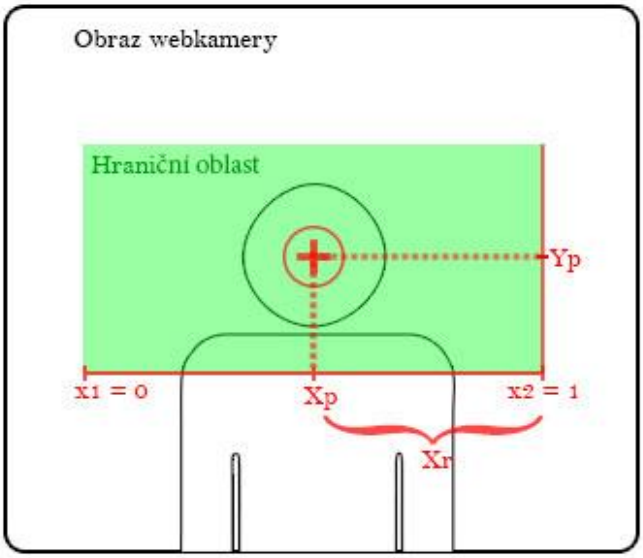
Obrázek 3.3 – Souřadnicový systém senzoru Kinect[7]

3.5.1 Kalibrace OpenCV souřadnic

Následující postup je nezávislý na typu souřadnicového systému, ze kterého převádíme.

První krok je vytýčení herního prostoru, ve kterém se uživatel bude pohybovat. Při snímání obrazu z kamery se musí obličej postupně dostat do všech hraničních pozic, pokaždé musí uživatel potvrdit pozici stiskem příslušné klávesy (konkrétně: „4“ – levá, „6“ – pravá, „8“ – horní, „2“ - spodní). Souřadnice těchto poloh se musejí pamatovat, následně se budou používat při kalibračním výpočtu. Algoritmus je navržený tak, aby před uložením zkontroloval, zda souřadnice dávají smysl (nelze například zaměnit levou a pravou hranici).

Následuje výpočet, který vezme aktuální souřadnici, kterou dodává webkamera a přepočítat jí na souřadnici pasující do právě vytvořeného hraničního prostoru. Cílem je dosáhnout toho, aby souřadnice x byla nulová, jestli se uživatel pohybuje před vytýčeným prostorem. Pokud je uživatel za označenou oblastí souřadnice by měla být rovna jedné. Herní prostor je tedy v rozsahu 0 až 1, toto číslo se násobí horizontálním rozměrem hrací plochy Pycanoidu. Výsledkem je souřadnice, která přesně zapadne do systému Pygame.



Obrázek 3.4 – Ilustrace vymezení hraničních bodů

Matematický postup:

$$Hranice\ z\ kalibrace = \{\ levá: x1, pravá: x2, horní: y1, spodní: y2\ \}$$
$$Aktuální\ (nekalibrovaná)\ pozice\ z\ kamery = (\ Xp, Yp\ )$$
$$Rozměr\ herní\ plochy = (X, Y)$$

Délka vyznačeného prostoru:

$$\boldsymbol{l = x2 - x1} \tag{3.5}$$

V případě, že se aktuální souřadnice pohybuje mezi x1 a x2 potřebuji informaci o tom, v jaké části to je:

$$\boldsymbol{Xr = x2 - Xp} \tag{3.6}$$

Toto číslo nyní stačí vydělit délkou, výsledkem je číslo v rozsahu 0 – 1:

$$\boldsymbol{Xk = \frac{Xr}{l}} \tag{3.7}$$

Kalibrovaná pozice se nyní rovná násobku velikosti herní plochy ve stejném rozměru.

$$\boldsymbol{Xp = Xk * X} \tag{3.8}$$

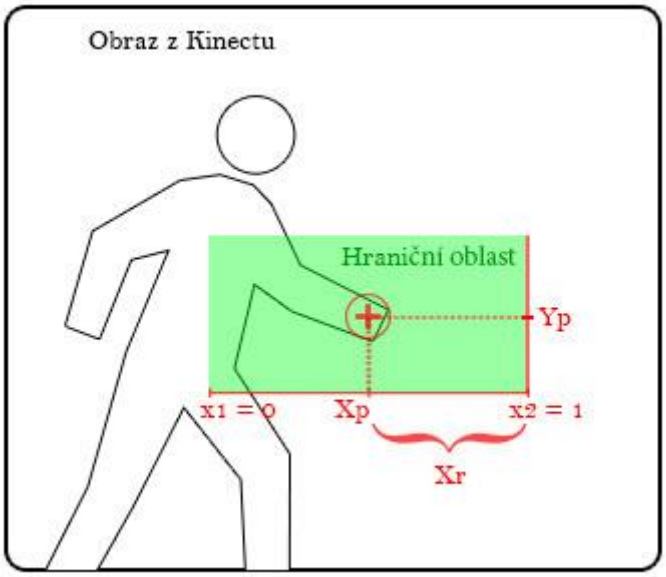
Když například bude mít herní plocha rozsah 0 až 1000 pixelů a Xk bude rovna 0.7, kalibrovaná hodnota bude odpovídat 700 pixelům.

3.5.2 Kalibrace senzoru Kinect

Tato kalibrace probíhá totožně jako v kapitole 3.5.1 s jedním rozdílem. Souřadnicový systém Kinectu inkrementuje souřadnici x směrem opačným než zrcadlově otočený obraz z webkamery. To lze napravit úpravou vzorce 3.7 :

$$\boldsymbol{Xk = 1 - \left(\frac{Xr}{l}\right)} \tag{3.9}$$

Při spuštění algoritmu kalibrace pro Kinect již není potřeba klávesnice. Program uživatele vyzve, aby vytýčil herní prostor pravou rukou. Kinect bude ruku snímat přibližně pět vteřin a mezitím si zapamatuje všechny souřadnice. Po ukončení si algoritmus sám vybere hraniční body, ze kterých bude provádět výpočet. Kalibrační výpočet nyní bere v potaz i souřadnici y. Jelikož uživatel nebude mít k dispozici myš, je vystřelování kuličky řízeno pohybem rukou vzhůru, respektive nad nejvyšší kalibrovanou souřadnici.



Obrázek 3.5 – Hraniční oblast pro Kinect

3.6 Vyhodnocení hráče

Po dohrání kola se vypočítává nahrané skóre. V potaz se bere počet vygenerovaných překážek, čistý herní čas a počet pokusů. Pro zajímavost se ještě pozoruje naježděná vzdálenost herní desky.

$$skóre = \frac{\frac{Počet\ překážek}{Počet\ pokusů}}{Čistý\ čas} * 1000$$

Rovnice 3.10 – Výpočet skóre v Pycanoidu

Vzhledem k tomu, že se počet překážek generuje náhodně, tak bylo potřeba tuto skutečnost nějak odrazit na výsledku. Čistým časem je myšlena doba, po kterou hráč odráží kuličku, pokud se kulička ztratí, čas se pozastaví, dokud jí uživatel znovu nevystřelí. To samé platí pro naježděnou vzdálenost, ta se výsledku ovšem neodráží. Vzdálenost se měří v pixelech.

{nick: '140510161615', number of blocks: 53, points: 271.0, takes: 2, time: 95.8, vzdalenost: 11016}

{nick: '140510161752', number of blocks: 57, points: 158.0, takes: 2, time: 176.88, vzdalenost: 15040}

{nick: '140510164122', number of blocks: 45, points: 113.0, takes: 3, time: 133.16,vzdalenost: 9853}

Ukázka 3.1 – Ukázka uložených dat a vypočteného skóre

3.7 Porovnání různých ovládání

Celkem jsou zavedeny tři typy ovládání hry: myš, webkamera a Kinect. Každé z nich vyžaduje různé řešení a implementaci.

Ovládání pomocí myši je v bezkontaktní hře prakticky nepoužitelné, ale při vývoji se mu nelze vyhnout. Většina vývoje probíhá právě při tomto ovládání, proto je perfektně odladěné a navíc má knihovna Pygame implementované funkce, které usnadňují práci s tlačítky.

Ovládání přes webkameru za pomocí OpenCV se příliš nehodí. Implementace je jednoduchá, ale výsledek není perfektní. Detekce není plynulá, pokud probíhá bez bližšího nastavení parametrů klasifikátoru. Ten se snaží detekovat obličej ve všech pozicích a velikostech, což chvíli trvá. Pokud se omezí maximální a minimální velikost hledaného obličeje, tak sice detekce funguje v reálném čase, ale sebemenší vybočení mimo parametry zapříčiní ztrátu obličeje. Pouhá změna v osvětlení scény způsobuje falešnou a neplynulou detekci. Při aplikaci na veřejné místo by tak stačila změna počasí a klasifikátor by nefungoval jako při prvotním nastavení. V potaz se také musí brát kvalita použité webkamery, na méně kvalitních modelech by se musel ještě implementovat takzvaný „face tracker“ (sledování obličeje), aby zabránil neustálému přeskakování detektoru do výchozí pozice. Po krátkém testu bylo rozhodnuto toto ovládání dále nerozvíjet.

Kinect je ze všech tří metod nejvýhodnější. Jeho implementace je sice velice složitá, vzhledem k tomu, že byl použitý model pro Xbox 360 a nikoliv pro Windows. K tomu aby do PC posílal souřadnice je třeba velké množství doplňujících knihoven. Informace které poskytuje, je třeba upravit do formátu, se kterým se dobře pracuje. Jakmile jsou souřadnice dostupné, spolupráce s Pycanoidem probíhá podobně jako při použití OpenCV. Nicméně klasifikátor Kinectu pracuje plynule a bez výpadků. Uživatel má volnost pohybu a přesto nedochází k falešné detekci. Kinect dokáže rozeznat více uživatelů najednou, ale sledovat jenom konkrétního, a tak se hodí pro veřejné prostranství. Osvětlení také nemá vliv, pokud by aplikace pracovala opravdu bez přestávky, tak není třeba zohledňovat počasí nebo denní hodinu.

4 Závěr

Cílem této práce bylo seznámit se s nástroji pro vizualizaci a zpracování obrazu v jazyce Python. Dále se seznámit s prostředky pro bezkontaktní komunikaci s počítačem. Pomocí těchto poznatků vytvořit hru použitelnou bez jakýchkoliv připojených periferií schopnou pracovat na veřejném prostranství v případném nepřetržitém provozu.

Ve druhé kapitole je popsaná nutná teorie k pochopení jednotlivých částí použitých při vývoji aplikace. V první části jsou popsány použité moduly knihovny Pygame a jejich rozložení v aplikaci. Kromě čistě teoretických informací o různých způsobech detekce v obrazu jsou v této kapitole probrána fakta o vzniku knihovny OpenCV a senzoru Microsoft Kinect. Metody detekce jsou probrány do větší hloubky pro lepší pochopení použitých technik.

Dále je navázáno třetí kapitolou, ve které jsou popsané praktické metody použité při vytvoření fyziky hry, řešení problému sladění několika různých souřadnicových systémů pomocí kalibrace a vysvětlení funkčnosti hry jako celku. Konec kapitoly je věnován prezentaci výsledků naměřených při použití aplikace a porovnání různých typů ovládání.

Aplikace dodržuje pravidla pro případnou expozici na veřejnosti. Jednoduchost herního cyklu pomáhá tomu, že není třeba mít hru neustále pod dohledem. Intuitivní ovládání pravou rukou nevyžaduje vysvětlování ani názorné ukázky. Fyzika hry funguje většinou dle očekávání, jelikož není použitá bodová kontrola kolize, ale toleranční oblast tak občas dochází k nepřirozeným odrazům. Práce na ovládání přes webkameru byla předčasně ukončena z důvodu malé použitelnosti. Webkamera neposkytuje potřebné vlastnosti pro bezchybné hraní na veřejném prostoru. Nicméně veškeré vytvořené metody jsou i přes to ve hře dostupné k případnému dalšímu vývoji.

Cílem optimalizace aplikace by mohlo být vylepšení načítání dat z Kinectu. Ovládání by dokázalo využívat prostorovou souřadnici k lepšímu odfiltrování okolního davu od uživatele. Nicméně propojení senzoru s počítačem podléhá

instalaci knihoven a balíčků, které nejsou dostupné pro operační systém Windows, což může stěžovat další vývoj nebo přenos aplikace. Ovládání pomocí OpenCV lze úplně vyřadit nebo ho modifikovat a využít jeho další funkce jako je například možnost chytré úpravy obrázků.

Literatura:

[1] **Microsoft Kinect Sensor and Its Effect** - Wenjun Zeng
http://research.microsoft.com/en-us/um/people/zhang/papers/microsoft%20kinect%20sensor%20and%20its%20effect%20-%20ieee%20mm%202012.pdf

[2] **Real-Time Human Pose Recognition in Parts from Single Depth Images** - Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio Richard Moore Alex Kipman Andrew Blake
http://research.microsoft.com/pubs/145347/bodypartrecognition.pdf

[3] **Face Detection using Haar Cascades**
http://docs.opencv.org/trunk/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

[4] **Using MS Kinect Device for Natural User Interface** - Petr Altman
ZČU 2013, Master Thesis

[5] **Pygame documentaion**
http://www.pygame.org/docs/

[6] **Integral Images**
http://www.nongnu.org/rapp/doc/rapp/integral.html

[7] **Microsoft Developer Network**
http://msdn.microsoft.com/en-us/library/hh973074.aspx

Zdroje:

[8] **Face detection with Nokia N900, using ROS and OpenCV**
http://sirloon.net/facedetection

[9] **Using The Kinect For Robotic Manipulation**
http://blog.robotiq.com/bid/40428/Using-The-Kinect-For-Robotic-Manipulation

[10] **Realtime Webcam Sudoku Solver –** Bojan Banko
http://www.codeproject.com/Articles/238114/Realtime-Webcam-Sudoku-Solver

[11] **Creative applications network**
http://www.creativeapplications.net/maxmsp/kinect-one-week-later-processing-of-cinder-maxmsp/

[12] **Kinect for blind**
http://www.zoneos.com/kinectfortheblind.htm

Seznam použitých rovnic:

Seznam ukázek:

Seznam použitých obrázků: