VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

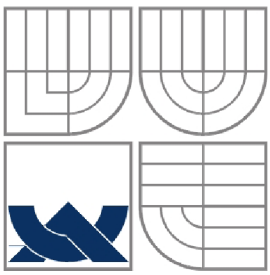
VÝUKOVÁ HRA NA PLATFORMĚ XNA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

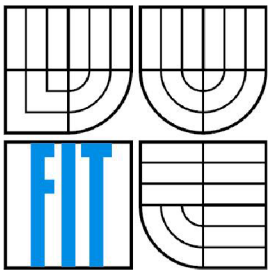
AUTOR PRÁCE
AUTHOR

Bc. LENKA VLKOVÁ

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝUKOVÁ HRA NA PLATFORMĚ XNA

EDUCATIONAL GAME FOR XNA PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LENKA VLKOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2010

Abstrakt

Cílem této práce je návrh a implementace výukové hry pro platformu XNA. Práce se zabývá vlastnostmi této platformy, především pak možností převodu hry na herní konzolu Xbox 360. Kromě návrhu a implementace 3D hry NanoHeal z oblasti zdravotnictví jsou zde popsány grafické efekty využité pro dosažení přitažlivého vzhledu hry. Jedná se o techniky nerealistického renderování, efekt zdůraznění hloubky ostrosti a částicové systémy. Klíčové charakteristiky výsledné hry byly zhodnoceny pomocí internetového dotazníku.

Abstract

This work deals with design and implementation of a game based on the XNA platform. It describes the platform and its possibilities of game development for both the PC and the Xbox 360 console. The implemented game is called NanoHeal and it is about the treatment of various health problems. The work also investigates graphic techniques such as non-photorealistic rendering, the depth of field effect and particle systems. These techniques are used to achieve the distinguished look of the game. Key features of the game were evaluated in the online questionnaire.

Klíčová slova

hra, vzdělání, XNA platforma, Xbox 360, shadery, nerealistická grafika, zdravotnictví

Keywords

game, education, XNA platform, Xbox 360, shaders, non-photorealistic rendering, health

Citace

Lenka Vlková: Výuková hra pro platformu XNA, diplomová práce, Brno, FIT VUT v Brně, 2010

Výuková hra pro platformu XNA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením Ing. Adama Herouta, Ph.D.

Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

……
Lenka Vlková
20. 5. 2010

Poděkování

Ráda bych poděkovala vedoucímu práce za ponechanou svobodu a rady při řešení projektu. Dále bych chtěla poděkovala svým rodičům za trpělivost, příteli za podnětné rady a inspiraci a oběma členům týmu Puellae za spolupráci a cenné tipy. Ještě bych chtěla poděkovat Ing. Rudolfu Kajanovi za ochotu a poskytnutí účtu pro testování na Xboxu 360.

Obsah

1 Úvod

Hry mají ve světě počítačů a informačních technologií již dlouhou historii. Zpočátku měli za úkol především odpočinek, relaxaci a zábavu. Postupně se však jejich funkce rozrostla a začaly se objevovat hry, které člověka nejen odvedly od každodenních starostí, ale také mu přinesly něco nového. Mohly mu nabídnout například emotivní zážitek, ale také vzdělání či nový pohled na svět. Myslím si proto, že se hry postupně vyrovnávají ostatním mediím, jakými jsou například filmy a knihy.

Počítačové a konzolové hry toho mnoho přináší do oboru počítačové grafiky. Každá hra se snaží být v něčem lepším než všechny ostatní a nejsnáze je možné potenciální hráče zaujmout především dech-beroucí grafikou. V současné době jsou již některé hry téměř nerozeznatelné od filmů a stále více a více se přibližují realističnosti skutečného světa. Existují však i hry, které na realističnost rezignují a snaží se spíše o umělecké ztvárnění herního světa. Tento způsob vykreslování herního světa se označuje jako nerealistické renderování (angl. Non-photorealistic rendering).

Ve své diplomové práci se budu zabývat nejen výukovými hrami, ale i některými technikami nerealistické grafiky. Jejich využitím je možné vytvořit graficky kvalitní a zajímavou hru i s omezenými prostředky. Dále budu popisovat vytvoření návrhu hry pro platformu XNA. Tato hra byla pracovně nazvaná NanoHeal a jejím cílem je rozšířit povědomí hráčů o světových problémech v oblasti zdravotnictví. Platforma XNA je revolučním počinem na poli konzolových her, protože umožňuje nadšencům vývoj na jednu z nejnovějších herních konzol – Xbox 360. Patří k technologiím, které se neustále rapidně vyvíjí, proto jsem se snažila čerpat nejnovější aktuality z internetových pramenů.

Hra NanoHeal byla vytvářena také za účelem účasti v soutěži Imagine Cup [5]. Tato soutěž se snaží upozornit na globální problémy lidstva a najít jejich řešení s pomocí moderních technologií. Projekt byl tvořen ve trojčlenném týmu, kde další dva členové měli na starosti vytvoření 2D herního obsahu a aplikace pro design úrovní hry. Mým hlavním úkolem v týmu byla implementace hry. Tématem soutěže stojícím i za myšlenkovým jádrem hry NanoHeal je motto: „Představte si svět, kde technologie pomáhá řešit ty nejpalčivější problémy“.

Během semestrálního projektu bylo vytvořeno jádro hry NanoHeal. Byla dokončena základní struktura menu a některé grafické efekty (renderování billboardů, nerealistické renderování hrdiny). Na tuto práci pak bylo navázáno a hra byla dokončena přidáním dalších grafických efektů, doplněním všech stavů hry, vylepšením logiky hry a dopracováním systému pro odblokovávání dalších úrovní. V tomto dokumentu jsem využila kapitolu 2 ze semestrálního projektu, mírně pozměněné kapitoly 3 a 4 a část kapitoly 5.

1.1 Členění do kapitol

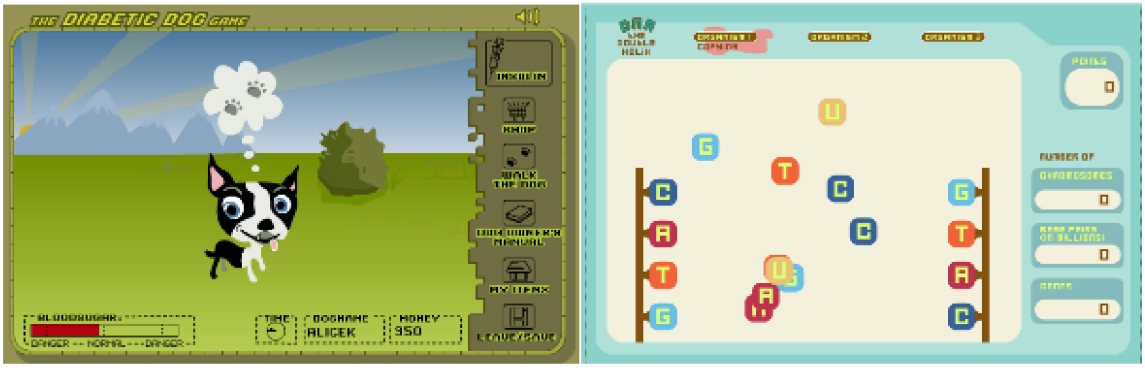
Ve druhé kapitole se nachází stručné shrnutí situace na poli výukových her, jejich využití a přínos. Také je zde rozebrána motivace pro tvorbu projektu a jeho cíle. V další kapitole jsou zmíněny nástroje a techniky pro tvorbu grafiky do hry a pro tvorbu samotné hry. Především zde popisuji platformu XNA. Stručně se zde také zabývám nerealistickou grafikou, která je podle mého názoru trochu neoprávněně ve stínu grafiky realistické. Ve čtvrté kapitole je popsána hra NanoHeal, její programový a grafický návrh a také v čem spočívá její poselství a naučnost. Definuji zde

i požadavky na její vytvoření. Další dvě kapitoly obsahují popis implementace zásadních prvků hry NanoHeal a použitých grafických efektů. V předposlední kapitole se věnuji přijetí hry uživateli vyhodnocením internetového dotazníku. Závěr obsahuje shrnutí výsledků diplomové práce a nastínění možného dalšího vývoje projektu.

2 Výukové hry pro PC i konzoly

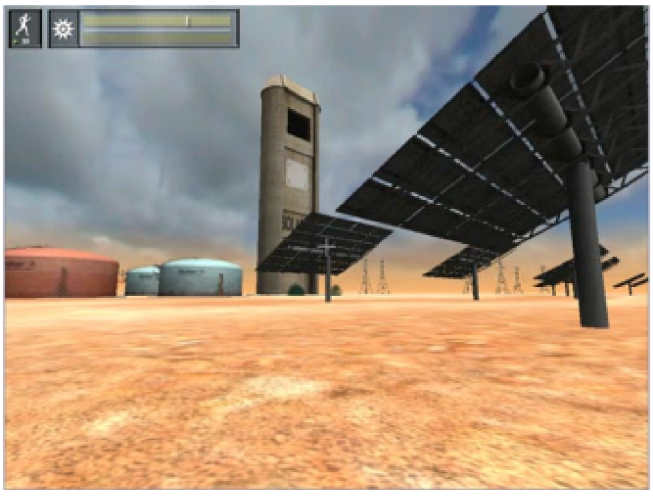
2.1 Současný stav

Na internetu i v prodejnách počítačových a konzolových her je možné najít mnoho výukových her a aplikací. Tradičně jsou populární hry na výuku cizích jazyků či matematiky. Existuje také mnoho webových portálů, které se na výukové hry přímo specializují [6]. V drtivé většině se jedná o 2D jednoduché hříčky, často vytvořené pro platformu Flash. Tyto hry bývají často primárně zaměřeny na mladší hráče, především děti.



Ilustrace 2.1: Ukázka výukových her z portálu Nobel Prize

Na portálu Nobelprize [7] se nacházejí hry, které interaktivní formou vysvětlují jevy, jejichž objevitelé obdrželi za výzkum či činnost v dané oblasti Nobelovu cenu (ilustrace 2.1 ukazuje hry vytvořené na základě obdržených cen v oblasti medicíny). Graficky výjimečným projektem je 3D hra Power Up firmy IBM [8], kde hráči kooperují a snaží se zachránit planetu Helios s pomocí obnovitelných zdrojů energie (viz Ilustrace 2.2).



Ilustrace 2.2: Ukázka ze 3D výukové hry PowerUp

2.2 Motivace a cíle

Při pohledu na současný stav výukových her je třeba říci, že tyto hry upřednostňují vzdělání před nápaditostí a propracovaným grafickým vzhledem. To je nejspíš jeden z hlavních důvodů, proč jsou tyto hry spíše stranou „běžných her“ v prodeji. Také hratelnost často nedokáže překročit hranice hříček zdarma, které jsou k dispozici na mnoha webových portálech (např. [9]). Dalším důvodem je fakt, že komerční hry v současné době tvoří týmy složené i z několika desítek členů, a tak je velmi těžké se jim (především graficky) vyrovnat.

Cílem této práce je vytvoření výukové hry, která využívá nejmodernější technologie vykreslování k vytvoření grafického vzhledu tak, aby zaujala nejen mladší, ale i starší hráče. Důležité je i nenásilné podání vzdělávacího faktoru. Uživatel by neměl zaregistrovat, že hraje výukovou hru, ale přirozeně vstřebávat informace. Měl by získat chuť vyhledávat další informace o zvoleném tématu.

Hra se snaží zájemce upozornit na nejzávažnější světové zdravotnické problémy, jakými jsou například malárie, riziko infarktu a podobně a poučit ho o jejich příčinách a následcích. Kromě samotného herního obsahu, kde se uživatel dostane do role nanorobota, který s těmito zákeřnými nemocemi bojuje, by měla hra obsahovat dějovou linku. Děj hry vykreslí nemoci v oblastech, pro které jsou tyto problémy typické (více v kapitole 4).

3 Nástroje a techniky pro tvorbu

Při tvorbě hry a dat pro hru je zapotřebí mnoho různých nástrojů. Nejvýznamnějším z nich je nejspíše programové jádro hry, herní engine. Herní engine by měl poskytovat:

- grafický engine, knihovny pro renderování 2D i 3D grafiky,
- zvukový engine, přehrávání 2D, 3D zvuků a hudby,
- kolizní systém, výpočet kolizí,
- fyzikální engine, výpočet kinematiky, dynamiky,
- podporu pro hru více hráčů,
- podporu pro skriptování herních událostí,
- nástroje pro editaci scény a další nástroje pro úpravu a práci s herními daty.

Existuje mnoho komerčních, ale i zdarma poskytovaných herních engine. Zdarma dostupné engine se však často věnují pouze některým částem komplexního herního engine (grafický engine, zvukový engine, engine na fyziku atd.).

Mezi nejznámější herní engine patří například komerční Unreal Engine, který používá mnoho nejnovějších her pro konzoly Playstation 3 a Xbox 360 [10]. Ze zdarma dostupných engine patří mezi nejpopulárnější grafický engine OGRE, grafický engine Irrlicht či fyzický engine Newton Game Dynamics.

Pro diplomový projekt byla použita platforma XNA verze 3.1 (více v podkapitole 3.1). Tato platforma sice usnadňuje vývoj her rozsáhlou nabídkou tříd pro práci s multimediálním obsahem, ale nedá se považovat za herní engine. Chybí např. práce s animacemi, složitější výpočet kolizí či funkce pro práci s fyzikou. Některé techniky využívané ve hrách (např. zobrazování billboardů – viz podkapitola 5.3) musely být doprogramovány.

3.1 Platforma XNA

XNA (rekurzivní zkratka „XNA is not acronymed“) obsahuje soubor nástrojů pro podporu vývoje her na platformy Windows, Xbox 360 a Zune (viz Ilustrace 3.1). Tyto nástroje vytvořila firma Microsoft, aby co nejvíce usnadnila vývoj pro tyto herní platformy a podpořila tím nejen profesionální herní vývojáře, ale i nadšence. Ti tak poprvé v historii dostali možnost vyvíjet legálně pro nejnovější herní konzolu.

Nástroje jsou k dispozici zdarma, ale pro vývoj na Xbox 360 (konkrétně pro testování, debugování na Xboxu 360 a distribuci přes obchod Xbox Live Marketplace) je nutné zakoupit prémiové členství do tzv. „Creator's club“ (čtyři měsíce stojí 49 amerických dolarů, roční členství pak 99 amerických dolarů) [1]. Pro studenty však existují speciální programy (např. DreamSpark), které umožňují získání členství pro spouštění a ladění aplikací na Xboxu 360. Tyto účty ale neumožňují prodej her přes Xbox Live Marketplace.

Ilustrace 3.1: XBox 360 s bezdrátovým ovladačem a ZUNE přehrávač

3.1.1 Součásti XNA

Klíčovým nástrojem XNA je vývojové prostředí XNA Game Studio. Jedná se o zásuvný modul pro integrované vývojové prostředí Microsoft Visual Studio, s jehož pomocí je možné snadno vyvíjet hry pro všechny podporované platformy XNA. Přehled verzí XNA Game Studia a nejdůležitějších přidaných funkcí naleznete v tabulce (Tabulka 1). Jak můžeme vidět v tabulce, tak vývojové nástroje platformy XNA se stále mění a vylepšují a nové verze přicházejí v přibližně jednoročních intervalech.

Verze	Datum vydání	Vlastnosti
XNA Game Studio Professional	nevydáno	Plánovaná verze pro profesionální vývojáře nakonec zcela ustoupila verzi zdarma
XNA Game Studio Express	Srpen 2006	Poprvé umožňuje nadšencům legálně vyvíjet na konzolu (Xbox 360), možnost výměny her skrze xna balík
XNA Game Studio 2.0	Prosinec 2007	Síťová podpora pro hru více hráčů, podporuje pro všechny verze Visual Studia 2005
XNA Game Studio 3.0	Září 2008	Podpora vývoje pro Zune, podpora C# 3.0, Visual Studio 2008
XNA Game Studio 3.1	Březen 2009	Přehrávání videa, možnost použití Xbox 360 avatarů ve hrách
XNA Game Studio 4.0 CTP	Březen 2010	Podpora pro vývoj na Windows Mobile Phone 7 platformu, Visual Studio 2010

Tabulka 1: Jednotlivé verze XNA Game Studia (zdroj [11])

Nejnovější verzí, která je v době psaní této práce k dispozici, je „technický náhled“ (nedokončená verze) XNA 4.0. Ta by měla dále rozšířit možnosti cílových platforem – a to o Windows mobilní telefony. Nástroje pro vývoj na tuto platformu byly totiž zahrnuty do XNA Game Studia spolu s dalšími vylepšeními.

XNA obsahuje také tzv. XNA Framework. Ten je založený na .NET Compact Frameworku a funguje na verzi CLR (common language runtime). Je tedy teoreticky možné programovat v jakémkoliv jazyce, který umožňuje .NET Framework. Oficiálně podporovaný je nicméně pouze jazyk C#. XNA Framework zastřešuje nízkoúrovňový kód různý pro jednotlivé platformy a umožňuje programovat s co největším znovupoužitím totožného kódu. Je však třeba brát v úvahu rozdílné hardwarové možnosti jednotlivých platforem (např. absence myši u Xboxu 360 a Zune či ukládání vlastních dat) a tyto případy zvlášť ošetřit. Vzhledem k tomu, že tento projekt je uvažován pouze pro platformy Windows a Xbox 360, budu v dalším rozboru odlišnou architekturu a vybavení platformy Zune a podporovaných mobilních telefonů zanedbávat.

XNA Framework využívají i další nástroje dodávané s XNA balíkem – například Cross- platform Audio Creation Tool pro tvorbu zvuků a hudby do her (viz kapitola 5.4). Poslední součástí balíku je XNA Build, který usnadňuje vytváření a kompilaci projektu pro různé platformy.

3.1.2 Podpora pro vývojáře

Kromě nástrojů, které usnadňují multiplatformní vývoj her, nabízí Microsoft i tzv. startovací balíky (angl. starter kits). Startovací balíky obsahují naprogramované jádro her různých žánrů (např. závodní hra, RPG, střílečka atd.). Jsou tedy skvělým zdrojem pro učení se programování pokročilých technik využívaných v jednotlivých typech her. Některé balíky jsou k dispozici pouze s placeným účtem pro vývoj, ale několik je k dispozici i zdarma (Platformer Kit, RPG Kit, Racing Game Kit, atd.).

Na oficiálních webových stránkách [1] je také možné najít mnoho příkladů zabývající se grafickými efekty či různými herními prvky (přechody mezi herními stavy, multiplayer, kolize, atd.). Kolem vývoje pro platformu XNA existuje velká a živá komunita, jejíž členové ochotně radí s různými problémy přes webové fórum.

3.1.3 Xbox 360

Platforma XNA je revoluční především proto, že dovoluje i neprofesionálním vývojářům tvořit hry pro herní konzolu – konkrétně herní konzolu od firmy Microsoft Xbox 360. Běžnou praktikou je totiž nutnost zakoupení drahého vývojového kitu dané konzole. Ten si mohou dovolit pouze větší vývojářské společnosti. Xbox 360 je herní konzolou sedmé generace. Tato generace bývá často označována jako „next-gen“ (herní konzole příští generace) a patří do ní ještě herní konzole od Sony Playstation 3 a konzole od Nintenda Nintendo Wii (viz Ilustrace 3.2).

Ilustrace 3.2: Playstation 3 v normální a slim verzi a Nintendo Wii (zdroj [19])

Vývoj pro konzolu (v tomto případě uvažuji Xbox 360) se oproti vývoji pro PC liší v mnoha aspektech. Pokud pomineme očividné aspekty (jako např. absence myši či speciální ovladač), musíme se vypořádat i s odlišnou architekturou. V době příchodu na trh (květen 2005) sice Xbox 360 představoval technologický vrchol, ale nyní již počítače opět pokročily a je třeba při vývoji na konzolu dělat oproti PC verzi jisté kompromisy (především při práci s pamětí). Specifikace konzoly Xbox 360 najdete v tabulce 2 [12]. Dalšími rozdíly se budu zabývat postupně v odpovídajících kapitolách.

Xbox 360 je v prodeji v několika verzích, které se však liší pouze změnou rozložení prvků na základní desce, velikostí pevného disku či podobnými změnami, které nesouvisí se samotným výpočetním výkonem konzoly.

CPU – Na zakázku vyrobený procesor od IBM Xenon	• tři symetrická jádra na frekvenci 3,2 Ghz • každé jádro obsahuje dvě hardwarová vlákna • jedna VMX-128 vektorová jednotka na jedno jádro • 128 VMX-128 registrů na jedno hardwarové vlákno • 1MB L2 cache paměť
GPU – na zakázku vyrobený od ATI Xenos	• 500MHz • 10MB vestavěné paměti DRAM • unifikovaná architektura shader jednotek • 48 paralelních dynamicky nastavovaných shader řetězců schopných práce s čísly s pohyblivou desetinnou čárkou
Paměť	• 512MB • 700MHz • unifikovaná architektura paměti
Úložiště	• vyměnitelný pevný disk (20GB) • dva vstupy na paměťové karty
Vstup/ Výstup	• až čtyři bezdrátové ovladače • tři USB 2.0 porty • DVD mechanika (podpora dvouvrstvých DVD)
Zvuk	• vícekanálový výstup – surround • podpora 48kHz 16ti bitového zvuku • 320 nezávislých dekompresních kanálů • přes 256 audio kanálů • 32bitové zpracování
Připojení	• ethernet port • možnost Wi-fi 802.11a, 802.11b a 802.11g

Tabulka 2: Základní specifikace Xboxu 360

3.2 Vývoj s XNA Frameworkem

Pro práci s XNA Frameworkem je nutné pochopit několik základních principů, které se liší od běžného programování pro PC. Jedná se především o práci s herními daty (zpracování pomocí tzv. content pipeline). Při tvorbě her v současné době je zásadní také práce se shadery, které již zcela nahradily fixní řetězec vykreslování scény. Nejprve je ale nutné popsat základní běh programu, který je při tvorbě s XNA Frameworkem pevně dán a připraven pro programátora ihned pro vytvoření nového projektu.

3.2.1 Herní smyčka

Základem každé hry je tzv. herní smyčka (angl. gameloop), která udává základní běh programu (hry). V každé hře se totiž střídají dvě fáze – aktualizace herního stavu a vykreslení scény. Aby hra korektně fungovala na různě výkonných sestavách, je nutné ošetřit správně načasování jednotlivých fází.

Při práci s XNA Game Studiem je tento problém vyřešen předem a programátor se může věnovat přímo programováním jednotlivých fázích a nemusí se zabývat jejich voláním. To zajišťuje XNA Framework. Základní běh XNA programu je vidět v následujícím pseudokódu:.

```
Inicializace aplikace
Načítání herního obsahu
Spuštění herní smyčky - dokud není konec hry
     Aktualizace herního stavu
     Vykreslení hry
Uvolnění herního obsahu
```

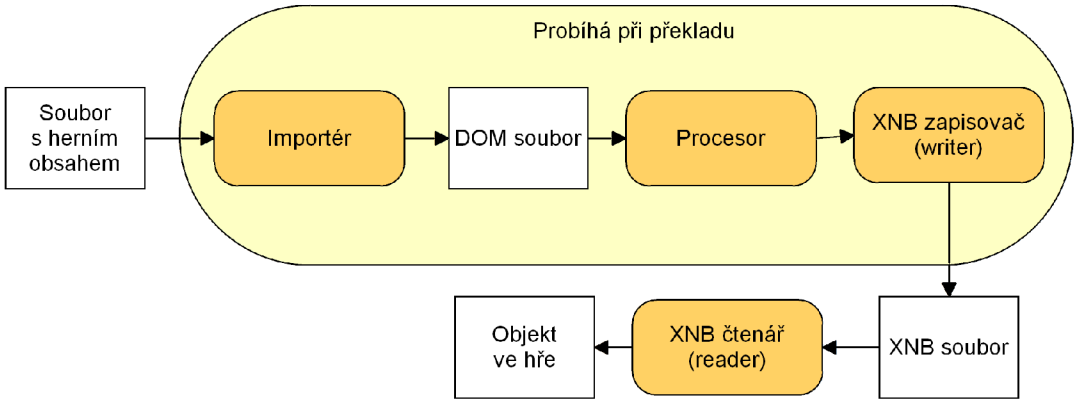
Nejprve nastává inicializace základních nastavení a po něm načítání herního obsahu (modelů, textur, zvuků, efektů, atd.). Poté program přechází do smyčky, v níž se střídá fáze aktualizace herního stavu (pohyb v herním světě, reakce na stisk tlačítka atd.) s vykreslováním scény (renderování, aplikace efektů, atd.). Po ukončení smyčky se uvolní načtený herní obsah a program je ukončen.

Časování herní smyčky je programově ovlivnitelné – je možné používat variabilní či pevné časování. Výchozí nastavením je pevné časování, při kterém se funkce aktualizace volá v pravidelných časových úsecích. V případě, že po funkci aktualizace zbývá nějaký čas do další aktualizace, program vyvolá i funkci pro vykreslení. V případě, že aktualizace trvala příliš dlouho, se vykreslení přeskočí. Během variabilního časování se obě funkce stále střídají nezávisle na uplynutém čase.

3.2.2 Content Pipeline – řetězec zpracování obsahu

Pro hry je typické, že kromě samotného programu obsahují i mnoho grafických, zvukových a dalších dat. Pro práci s těmito daty (označované jako herní obsah) se v XNA Frameworku používá tzv. content pipeline.

Data při běhu programu nejsou načítána ve svých nativních formátech (např. textury nejsou ve formátech jako .jpeg, .png, .tga, atd.). Již při překladu se tato data zpracují a převedou do binárního formátu .xnb. Důvodem tohoto zpracování je především kombinace omezené paměti a absence virtuální paměti na Xboxu 360 a faktu, že při zpracovávání těchto dat je potřeba velké množství dočasné paměti. Veškeré zpracovávání načítaných dat tak proběhne vždy při překladu pod systémem Windows.



Ilustrace 3.3: Řetězec zpracování herního obsahu

Z obrázku (viz Ilustrace 3.2 – převzato z [13] a upraveno) je patrné, jak je daný soubor postupně zpracováván. Nejprve se načte importérem pro daný typ souboru a zkonvertuje do objektu DOM (dokument object model). Nativně XNA obsahuje vestavěné importéry pro velké množství 2D obrazových formátů, 3D formáty .x, .fbx a další. Následuje zpracování procesorem, ve kterém může probíhat úprava načtených dat. Nakonec se soubor uloží do formátu xnb, jenž je pak při běhu programu možné efektivně načítat (xnb writer a reader).

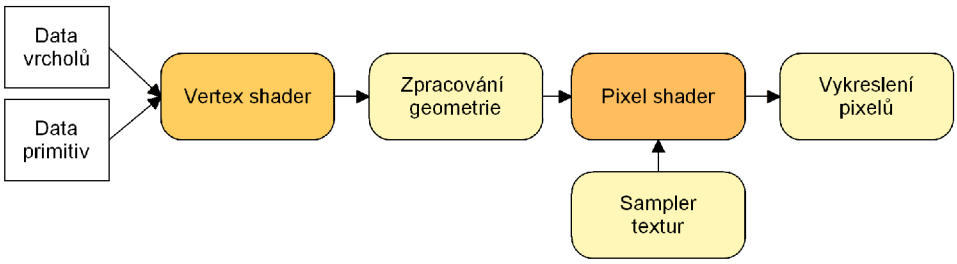
XNA umožňuje nahradit kteroukoliv fázi svým vlastním kódem. Tímto způsobem je možné přidat do her vlastní formáty herních dat či pouze rozšířit řetězec o zpracovávání normálových textur (vlastní procesor) a podobně.

Od verze 3.1 obsahuje Game Studio automatickou XML serializaci souborů s využitím mechanismu reflexe, a proto není nutné při tvorbě vlastního typu zpracovávaných dat psát vlastní konvertor z a do formátu xnb. Vzhledem k použití reflexe toto načítání může být v některých situacích pomalejší, a proto je někdy výhodnější využití vlastního konvertoru [13].

3.2.3 Práce se shadery

Vzhledem k tomu, že Xbox 360 podporuje pouze DirectX 9, tak i vývoj v XNA Frameworku umožňuje podporu nejvýše shader modelu 3,0. Z toho, že XNA Framework je prakticky „zabalené“ DirectX 9 plyne i to, že je zde možné psát shadery (programy pro grafickou kartu) v jazyce HLSL.

Shadery (označované jako efekty) se ukládají a používají stejným způsobem jako ostatní herní data. Pro jednoduché efekty je možné použít vestavěný `BasicEffect`, který provádí potřebné operace pro vykreslení objektu (transformace) a umožňuje mnoho nastavení pro osvětlení a podobně.



Ilustrace 3.4: Vykreslovací řetězec v XNA

Na obrázku 3.2 je možné vidět vykreslovací řetězec XNA, přičemž je možné s pomocí efektů naprogramovat vlastní zpracování vrcholů a pixelů (vertex shader, pixel shader). Ostatní části je možné ovlivňovat pouze různými nastaveními (např. nastavení ořezávání u zpracování geometrie).

3.3 Tvorba 2D a 3D grafiky pro hru

Volba software pro tvorbu grafického obsahu do her a jeho formátu může být závislá na možnostech používané platformy. Díky široké formátové podpoře XNA Frameworku jsem u tohoto projektu téměř žádná omezení nezaznamenala. Výběr probíhal především na základě vlastnostech jednotlivých software. Za klíčovou vlastnost byla považována možnost používat daný software zdarma.

3.3.1 2D grafika

Pro úpravu textur byly využity především programy Gimp a Paint.NET. Tyto aplikace jsou k dispozici zdarma i pro komerční využití a nabízí široké spektrum funkcí pro práci s obrázky. Jako hlavní formát pro textury byl zvolen na základě příznivých vlastností formát .dds. Ten se široce používá právě ve hrách. Díky speciální kompresi, kterou je grafická karta schopna dekomprimovat bez nutnosti pomalé komunikace s CPU, nezabírá příliš místa v paměti a jeho zpracování je rychlé. Formát DDS umožňuje také přímo v textuře ukládat mipmapy a je XNA Frameworkem jako původně formát vyvinutý pro DirectX nativně podporován.

3.3.2 3D grafika

Modely pro hru byly tvořeny s pomocí aplikace Blender. Blender je nejlepší aplikací zdarma dostupnou i pro komerční využití. V komerční sféře se v většinou používají aplikace Autodesku 3D Studio Max (například společnost Blizzard) či Maya (například společnost Square-Enix). Blender však nabízí uspokojivou implementaci všech funkcí potřebných pro tento projekt. Je v něm možno model nejen vytvořit, ale i na-texturovat a na-kostit pro možnost animace a případně i na-animovat.

Vzhledem k tomu, že XNA umí nativně pouze formáty .x a .fbx, bylo nutné využít exportéru z nativního formátu Blenderu .blend. Vestavěný .fbx exportér ale není se specifikací v XNA

kompatibilní a při exportu do formátu .x jsem se potýkala s problémy kvůli přechodu mezi levotočivým a pravotočivým systémem souřadných os.

Řešením je speciálním exporter pro Blender. Je totiž vytvořený přímo pro použití v kombinaci s XNA. Umožňuje korektní export nejen vertexů s texturami, ale i animací [14]. Pro správný export na-kostěných modelů je nutné dodržet fakt, že každý vrchol musí mít přiřazenou některou z kostí i přes to, že by to nebylo nutné (nebyl by součástí animace) [3].

3.4 Nerealistická grafika

Již mnoho let se počítačová grafika stále více a více snaží přiblížit k realitě a vypadat jako skutečný svět. Z obrázku (Ilustrace 3.4 vlevo) téměř nelze poznat, že se jedná pouze o hrdinu hry renderované v reálném čase. Trochu ve stínu drastického přibližování se k realitě existuje ještě jeden směr počítačové grafiky a tím je právě nerealistická grafika. Jedná se o stylizované vykreslování, které se místo realitě snaží přiblížit spíše různým uměleckým směrům [15], [16] a ručnímu kreslení. Existuje proto mnoho různých typů nerealistického renderování (malířské techniky, črtání).

I přes to, že o nerealistické grafice není příliš slyšet, má tento obor bohatou tradici a již od roku 2000 se koná sympózium [17], kde se sdílí nově dosažené znalosti a zjištěné techniky. I ve hrách lze najít bohaté využití nerealistické grafiky (viz Ilustrace 3.4 vpravo). Díky ní je možné hře zajistit jedinečný vzhled a upoutat hráče i bez nutnosti pořizování drahého grafického engine, který podporuje všechny možnosti realistického renderování.

Ilustrace 3.5: Hra Uncharted 2 pro Playstation 3 využívající realistické renderování a hra Valkyria Chronicles pro Playstation 3 využívající nerealistické renderování (převzato z [18])

Tento projekt byl také navržen s cílem využít technik nerealistické grafiky – zvláště pak tzv. cartoon renderingu – k dosažení jedinečného a poutavého vzhledu.

3.4.1 Vykreslování ve stylu cartoon

Vykreslování ve stylu cartoon (cartoon rendering) je jednou z technik nerealistické grafiky. Často bývá označováno jako Cel-shading. Jedná se o techniku, která renderuje 3D scénu tak, aby působila jako nakreslená rukou ve stylu 2D kreseb. K dosažení tohoto efektu se kvůli možnosti zpracování v reálném čase využívají především shadery.

Pro požadovaný efekt je třeba zajistit vykreslení obrysu 3D modelu (často černou) linkou a „kvantifikované stínování“. Pro vykreslení obrysu je možné využít při renderování například normál modelu v kombinaci s hloubkovou mapou obrazu či tradiční algoritmy detekce hran (např. Sobelův detektor). Při výpočtu osvětlení je pak nutné zamezit plynulému stínování a vykreslovat stíny kvantifikovaně. Konkrétní technika použitá pro vykreslování ve stylu cartoon v tomto projektu je vysvětlena v kapitole 6.2.

4 Výuková hra NanoHeal - návrh

V této a následujících dvou kapitolách se budu zabývat hlavním cílem diplomové práce. Tím je návrh a implementace výukové hry pro platformu XNA. Hra byla nazvána NanoHeal (nano léčení), protože hráč se dostává do role nanorobota, který léčí nejzávažnější a nejproblematičtější nemoci dnešní doby. Děj hry NanoHeal se odehrává po celém světě a hráč se tak dostává do krevního řečiště různých pacientů, kteří trpí nemocemi typickými pro danou oblast světa. Jeho úkolem je dostat se až do centra působení baktérií a vyléčit nemocného sbíráním různých předmětů a buněk nalézajících se v krevním řečišti.

4.1 Požadavky na hru

Ještě před zahájením samotné práce na projektu bylo nutné specifikovat základní požadavky na vytvořenou hru. Tyto požadavky se dají rozdělit na požadavky na:

- grafickou podobu hry,
- herní prvky,
- naučnost.

Všechny požadavky a plány byly sepsány do tzv. design dokumentu, který se postupně upravoval a podle kterého se vývoj projektu řídil. Je k dispozici na DVD příloze k tomuto dokumentu.

4.1.1 Grafické požadavky

Vzhledem k tomu, že jedním z cílů je vytvoření hry přibližující se úrovni komerčních titulů, jsou požadavky na grafiku hry relativně vysoké. Vytvoření kvalitních 3D modelů pro kompletní hru, která obsahuje několik herních oblastí, je však velmi náročné na čas i lidské zdroje. Hledal se proto způsob, jakým dosáhnout co nejvyšší kvality vhodnou kombinací 3D a 2D grafiky.

Finální grafické požadavky byly určeny následovně:

- 3D herní prostředí,
- 3D hrdina,
- animovaní nepřátelé a prvky herního prostředí ve 2D,
- nerealistické renderování 3D modelů s pomocí shaderů,
- post-processing scény s pomocí shaderů,
- částicové efekty pro oživení scény,
- interaktivní a graficky nápadité menu,
- příběh podaný formou videa či obrázků.

4.1.2 Herní prvky

Nejdůležitějším faktorem, který rozhoduje o úspěchu a popularitě hry, je hratelnost. Tedy to, jak je hra zábavná při hraní a zda-li hráče dostatečně motivuje k dalšímu hraní. Požadavky na herní prvky byly s ohledem na možnost kompletního dokončení projektu v omezeném čase formulovány následovně:

- omezený pohyb v 3D prostředí,
- fixní kamera,
- interakce s nepřáteli a prostředím,
- sbírání surovin,
- členění hry do jednotlivých herních úrovní,
- vzrůstající obtížnost a herní mechaniky,
- udržování herního skóre.

4.1.3 Naučnost

Klíčovým problémem bylo i správné zakomponování naučnosti do hry tak, aby neodradila hráče přílišným poučováním. Proto bylo rozhodnuto o naučnosti v co nejméně invazivní formě – skrze příběh, prostředí a databázi s informacemi. Požadavky byly stanoveny na:

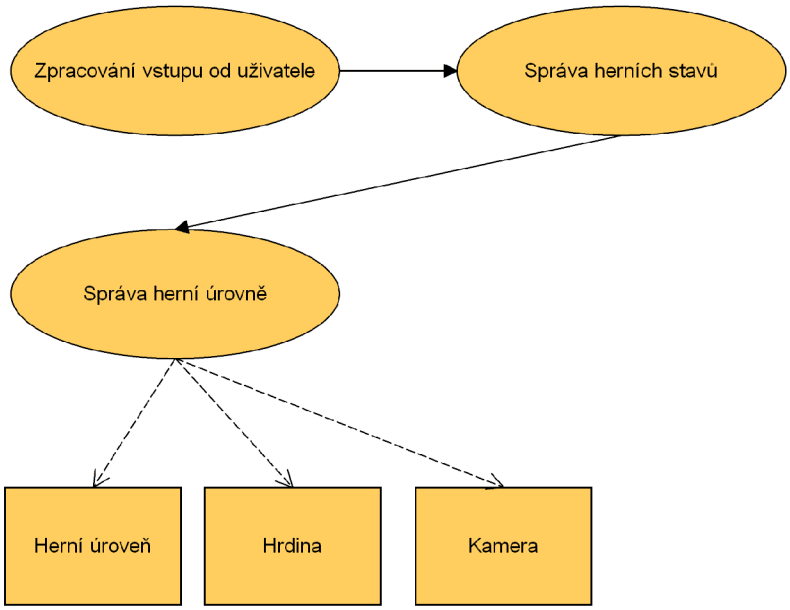
- databáze s podrobnými informacemi o nemocích a jejich výskytu,
- co nejrealističtěji ztvárněné krevní řečiště,
- příběh, který vykreslí pravdivě a zábavně situaci v dané části světa.

4.2 Programový návrh aplikace

Při návrhu jádra hry jsem postupovala kombinovaně metodou shora-dolů i zdola nahoru. Nejprve bylo nutné navrhnout grafické jádro, které zobrazuje herní prostředí a hrdinu. Poté jsem postupovala zdola přidáváním jednotlivých částí, jakými jsou například zpracování uživatelského vstupu, třídy na načítání herní úrovně a nastavení, třída pro manipulaci s kamerou a další.

Základní struktura programu je nezávisle na konkrétním rozdělení do tříd zobrazena na obrázku 4.1. Jakmile se spustí herní smyčka, předává se řízení aplikace Manažeru herních stavů. Herní stav udává, v jaké fázi se hra nachází. Zda-li je hráč v menu, v nastavení či přímo ve hře. Na základě herního stavu se odvíjí další činnosti programu. Manažer dostává informace o vstupu od hráče a předává je dál.

Ještě před spuštěním herní smyčky se načítá perzistentní herní obsah, který je třeba po celou dobu běhu udržovat v paměti – tím je například nastavení hry. Další načítání herního obsahu pak probíhá v rámci každého herního stavu. Vykreslování samotné hry a výměnu dat pro jednotlivé úrovně pak má na starosti Manažer herní úrovně. Ten se kromě herních dat stará i o vykreslování a aktualizaci hry. Obsahuje instance současné herní úrovně, hrdiny, kamery a dalších objektů potřebných pro hru.



Ilustrace 4.1: Základní struktura projektu NanoHeal

4.2.1 Herní stavy

V každé hře se nachází jisté herní stavy, jakými jsou například hlavní menu, nastavení, samotná hra a další. I pro projekt NanoHeal bylo nutné navrhnout schéma těchto herních stavů a jak se mezi sebou budou vzájemně měnit. Dá se říci, že herní stav reprezentuje jednu herní obrazovku. Diagram všech herních stavů je možné vidět na obrázku 4.2. NanoHeal je kompletní hrou se všemi tradičními náležitostmi, a tak je diagram poměrně rozsáhlý.

Po spuštění se aplikace přepne do stavu Intro, ve kterém se zobrazí logo týmu tvůrců a úvodní logo hry. Poté se hra přesune do hlavního menu, kde může uživatel volit další postup. V Databázi bude možné prohlížet postupně odblokovávané záznamy o jednotlivých nemocích a o látkách a buňkách v lidském těle.

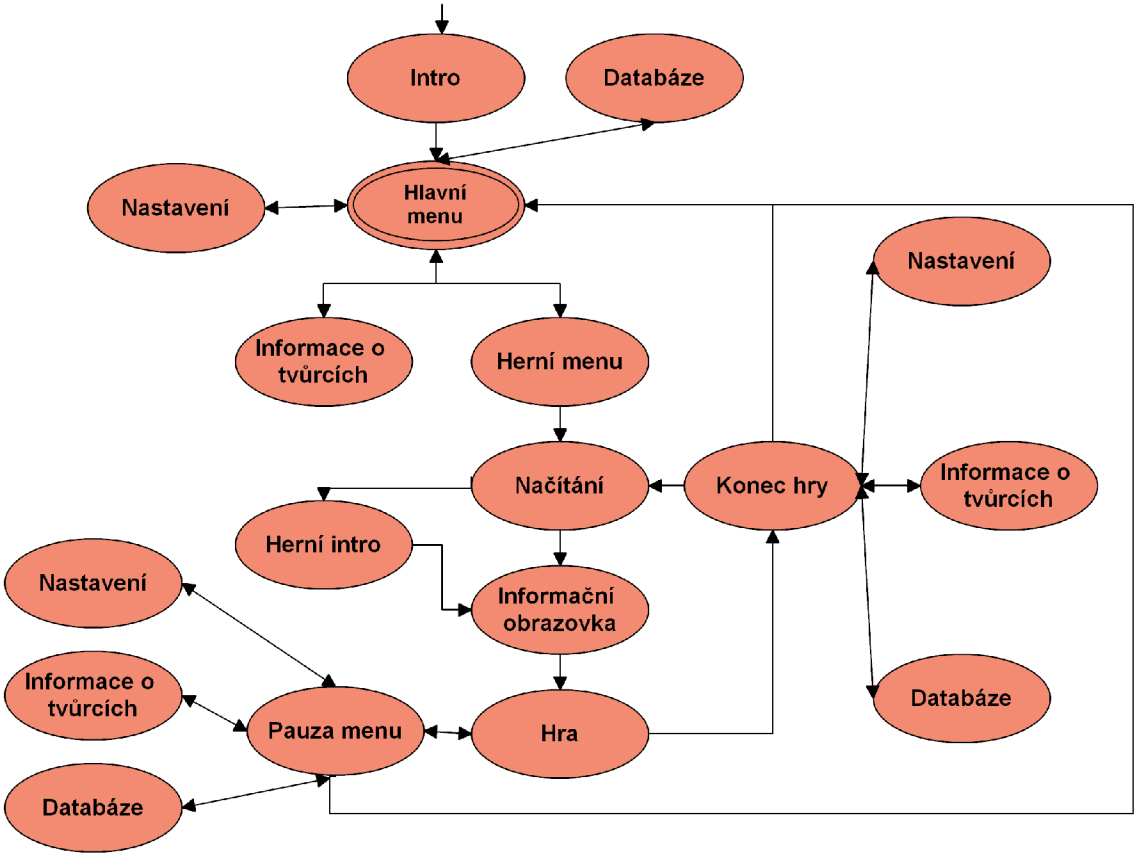
V nastavení bude mít hráč možnost měnit parametry zobrazení aplikace (rozlišení, přepnutí na celou obrazovku) či ovládat hlasitost zvuků a hudby. Posledním ale neméně důležitým stavem přístupným z hlavního menu je obrazovka s informacemi o tvůrcích.

Po přesunu do herního menu hráč zvolí požadovanou úroveň hry a tím se dostane do stavu Načítání. V tomto stavu probíhá načítání herních dat pro danou úroveň. Následuje herní intro (pouze pro některé úrovně) a zobrazení tzv. Informační obrazovky. Ta ukazuje podmínky pro splnění dané herní úrovně.

V herním stavu je pak možné hru kdykoliv pozastavit. Z Pauza menu bude mít hráč přístupné stejné položky jako z hlavního menu. Pokud chce hráč ukončit hru jiným způsobem než vítězstvím či prohrou v dané úrovni, je to možné opět v pauza menu.

Pokud hráč dohraje herní úroveň (úspěšně či neúspěšně), bude mít ze stavu Konec hry možnost návratu do hlavního menu hry či pokračování ve hře v další (případně znovu stejné při neúspěšném

dohrání) herní úrovni. Změna nastavení či prohlédnutí databáze a informací o autorech je také možná z tohoto stavu.



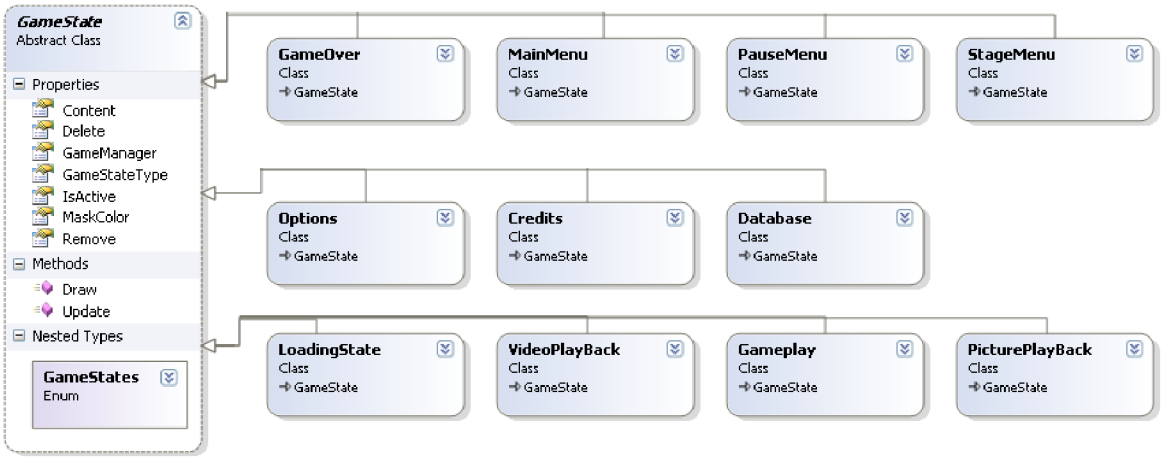
Ilustrace 4.2: Přehled stavů hry

5 Implementace hry

V páté kapitole této práce se budu zabývat negrafickou částí implementace hry. I přesto, že grafika hry je klíčová pro vytvoření zájmu a nalákání hráčů, tak ostatní části - jakými jsou například herní logika či zvuky, jsou neméně důležité pro celkovou zábavnost a zajímavost hry.

5.1 Implementace herních stavů

V podkapitole 4.2.1 byly popsány jednotlivé herní stavy. Jejich správné střídání a aktualizace jsou klíčové pro správný chod hry. Každý herní stav je odvozen od abstraktní třídy `GameState` (Ilustrace 5.1). Tato třída obsahuje všechny data potřebná pro všechny herní stavy. Jedná se například o stav aktivity (`IsActive`), informace o tom, zda-li je stav již ukončen a čeká na odstranění (`Remove, Delete`) či vlastní manažer herního obsahu (`Content`) pro včasné odstraňování nepotřebných dat z paměti.



Ilustrace 5.1: Herní stavy ve hře NanoHeal

Třída `GameState` také zajišťuje plynulý přechod mezi jednotlivými herními obrazovkami. Pokud je totiž herní stav označen k odstranění, tak není odstraněn ihned, ale postupně zprůhledňován a až se hodnota alpha kanálu dostane na maximum (255), tak je teprve označen k úplnému odstranění. Stejně tak při aktivaci herního stavu se objevuje postupně. Zprůhledňování a zviditelňování je zcela v režii základní třídy `GameState`. Odvozené třídy jednotlivých stavů totiž k vykreslování využívají její `MaskColor` proměnnou. Tato proměnná se pak aktualizuje při aktivaci a deaktivaci stavu skrze aktualizační funkci (`Update()`) základní třídy, kterou je třeba v aktualizační funkci každé odvozené třídy volat.

Mezi proměnné třídy `GameState` patří také manažer obsahu `Content` (třída `ContentManager`). Ten se v XNA používá pro spravování načteného herního obsahu. Díky tomu, že každý herní stav si vytvoří svůj vlastní manažer, je možné herní obsah uvolňovat postupně a šetřit tak s dostupnou pamětí programu.

Jednotlivé herní stavy se ukládají do zásobníku, který zpracovává Manažer herních stavů (viz Kapitola 4.2). Při každé aktualizaci hry se pak aktualizuje stav na vrcholu zásobníku a vykreslují se dva herní stavy na vrcholu zásobníku. V případě, že je nejhornější stav připraven k odstranění, je odstraněn ze zásobníku a na jeho místo se dostane a je aktivován následující herní stav.

5.1.1 Stavy pro zobrazení animací

Vzhledem k požadavkům na podání příběhu (viz 4.1.1) jsem vytvořila systém pro přehrávání videí. Od verze 3.1, která byla při tvorbě tohoto projektu použita, XNA přímo podporuje přehrávání videa ve formátu .wmv. Vestavěná třída `VideoPlayer` pro přehrávání videí byla zakomponována do herního stavu `VideoPlayBack`. Tento herní stav umožňuje přehrát libovolné podporované video. Při vytváření stavu je také možné nastavit možnost přeskočení videa stisknutím klávesy.

Na základě zpětné vazby od uživatelů bylo rozhodnuto, že herní příběh (původně podávaný formou videí) bude působivější v případě přímé interakce hráče. Ten si sám může rozhodnout, kdy se přesune na další snímek animace. Tuto funkci plní v programu herní stav `PicturePlayback`.

Tento stav také umožňuje přeskakování animace (to je vhodné v případě hraní stejné úrovně po několikáté). Na další snímek je možné se přesunout stisknutím klávesy. Mezi jednotlivými snímky probíhá prolínání za pomoci alfa kanálu. V případě dlouhé nečinnosti je uživatel upozorněn na stisknutí klávesy blikající šipkou.

5.1.2 Stav Načítání

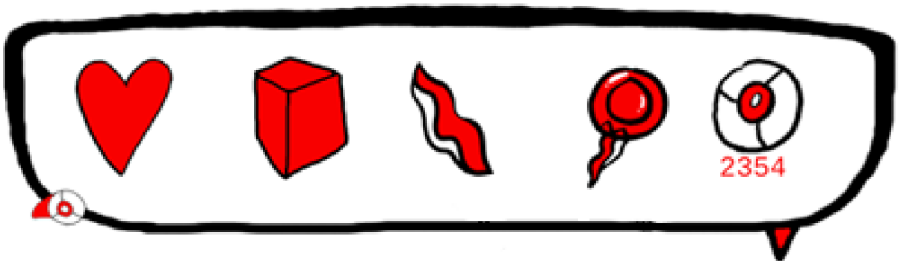
Vzhledem k tomu, že načítání herních dat pro celou herní úroveň může trvat nezanedbatelnou dobu (až několik sekund), bylo třeba věnovat zvláštní pozornost stavu Načítání (třída `LoadingState`). Během této doby je totiž třeba hráče něčím zaujmout a dát mu vědět, že hra normálně pokračuje a funguje. Proto bylo potřeba vytvořit vlákno na pozadí, které bude během načítání dále vykreslovat a aktualizovat herní obrazovku. Po skončení načítání herních dat se tomuto vláknu pošle signál pro ukončení a program dále pokračuje.

Tato doba, kdy hráč nemůže nijak interagovat s aplikací, je také vhodná pro vypisování různých informací a tipů pro hru. Vybrala jsem tedy základní informace o nemocích, s nimiž se hráč ve hře postupně setkává. Jsou jimi malárie, infarkt a AIDS. Jednotlivé nemoci jsou pro pestrost hry výrazně barevně odlišeny, a tak i nahrávací obrazovka se mění na základě použitých barev (Ilustrace 5.2).

Ilustrace 5.2: Nahrávací obrazovka pro druhou nemoc infarkt

5.1.3 Stav Hra

Stav Hra má na starosti nejdůležitější část aplikace. Skrze `LevelManager` (manažer herní úrovně) provádí aktualizaci a vykreslování aktuální herní úrovně. Sám pak vykresluje menu ve hře (viz Ilustrace 5.3), které hráči ukazuje (zleva) zdraví pacienta, pohon (cukr), sbírané prvky a herní skóre. Informace o kolizích s prvky v úrovni se zjišťují v objektu třídy `Stage`, jejíž instanci vlastní také `LevelManager`. Podrobněji o kolizích a herní logice bude popsáno v kapitole 5.3.

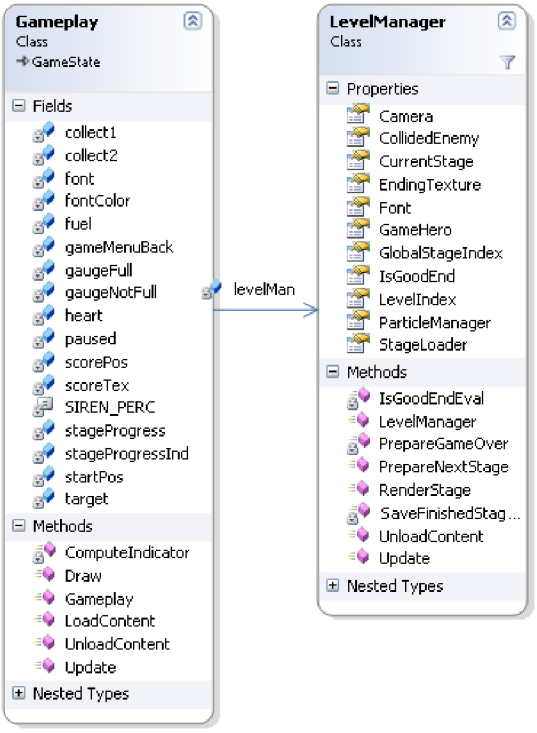


Ilustrace 5.3: Menu ve hře pro nemoc malárie

Třídy pro správu hry můžete vidět na následující ilustraci 5.4. `Gameplay` (stav Hra) obsahuje struktury pro vykreslení jednotlivých herních ukazatelů (`collect1`, `collect2`, `fuel`, `heart`) a také funkci pro výpočet kolik procent z textury s plným ukazatelem se má vykreslit – `ComputeIndicator()`. Dále obsahuje také instanci třídy `Target`, která hráči pomáhá určit který z objektů je pro něj škodlivý a který by měl sbírat zobrazením pomocné šipky nad nejbližším objektem (více o herních mechanikách bude uvedeno v další podkapitole).

`LevelManager` obsahuje především instance kamery, hrdiny – nanorobota, aktuální herní úrovně a manažera částicových systémů (viz kapitola 6.4). Dále sleduje informace o konci hry a předává stavu Hra informaci v případě, že je třeba přejít do jiného herního stavu. `LevelManager` také spravuje přechody do další úrovně s pomocí funkcí `PrepareNextStage()`. Jeho důležitou

funkcí je i renderování 3D scény včetně zpracování vykreslené scény efektem. Tomuto se budu podrobněji věnovat v kapitole 6.

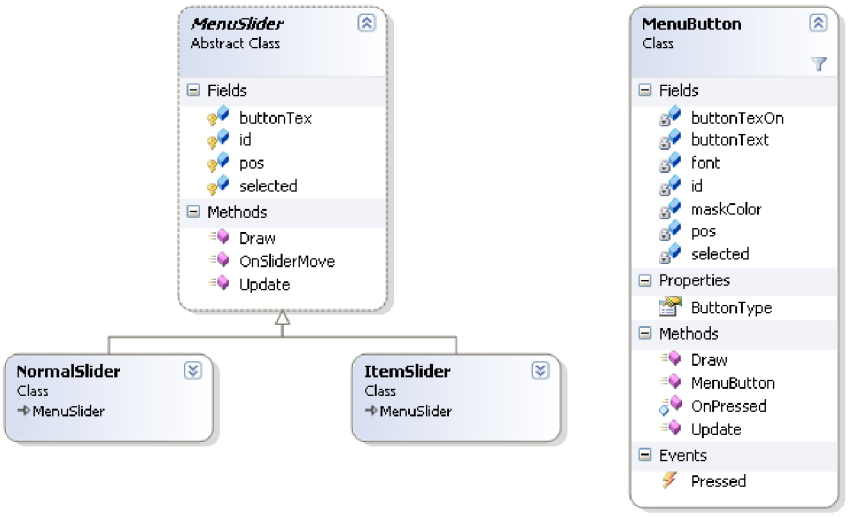


Ilustrace 5.4: Třídy pro správu stavu Hra

5.1.4 Ostatní herní stavy

Mezi ostatní herní stavy patří především stavy různých menu (nastavení, hlavní menu, menu výběru herní úrovně a podobně) a databáze. Pro obrazovky bylo nutné naprogramovat prvky grafického uživatelského rozhraní. Jejich diagramy tříd můžete vidět na ilustraci 5.5.

Nejvíce používaným prvkem je tlačítko. Při stisku tlačítka se vyvolá událost `Pressed`, která vyvolá nastavenou funkci pro určité tlačítko. Třída tlačítka dále obsahuje klasické informace jako například texturu pro označení tlačítka, pozici, identifikaci tlačítka a tak dále.



Ilustrace 5.5: Třídy prvků uživatelského rozhraní

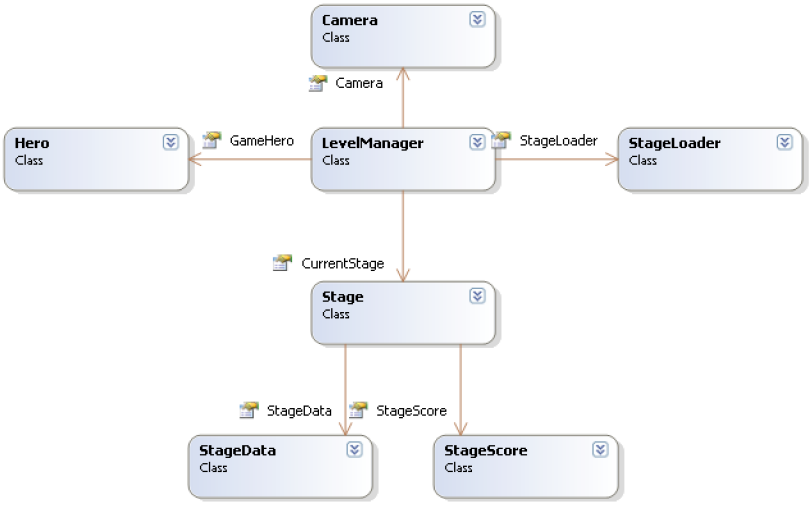
Dále bylo třeba vytvořit posuvníky. Rozhodla jsem se pro vytvoření dvou typů posuvníků. Jedním je klasický posuvník pro nastavování např. hlasitosti. Druhým je pak speciální posuvník, který umožňuje posouvat ne plynule, ale skokově. Vzhledem k absenci myši na Xboxu bylo ovládání koncipováno pouze pro gamepad a klávesnici. Právě kvůli snadnému ovládání gamepadem a šipkami na klávesnici se tento posuvník ukázal jako vhodný k přepínání nastavení jako například grafické rozlišení hry či přepnutí na celou obrazovku (viz Ilustrace 5.5).

Ilustrace 5.6: Posuvník pro nastavení obrazu na celou obrazovku

5.2 Herní logika

V podkapitole 5.2 se budu zabývat implementací samotného jádra hry, tedy herní logiky. Budu se věnovat především základnímu principu hry NanoHeal a třídami, které je zpracovávají. Jedná se především o třídu herního hrdiny – `Hero`, o třídu kamery – `Camera` a dále pak o třídu `Stage`, která spravuje veškeré objekty v herní úrovni a provádí například výpočet kolizí.

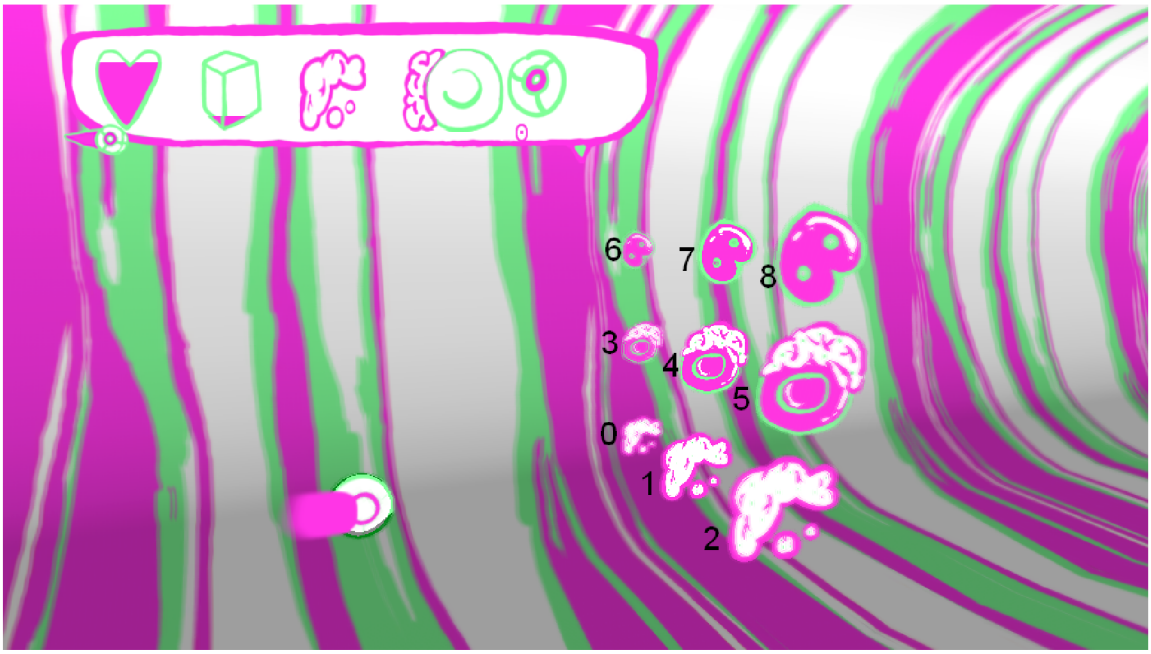
Jak jsem již zmínila v předchozí kapitole, tak správu herní úrovně provádí třída `LevelManager`, jejíž instanci vlastní vždy stav Hra. Ten jí poskytuje informace o uživatelském vstupu a volá její vykreslovací a aktualizační funkci. V nich se pak volají odpovídající funkce tříd potřebných pro výpočet herní logiky (viz Ilustrace 5.7).



Ilustrace 5.7: Třídy pro logiku hry

5.2.1 Princip hry

Základním principem hry je interakce s objekty v krevním řečišti. Tím se hrdina hry nanorobot může pohybovat omezeně na devíti „kolejích“. V ose X a Y jsou vždy tři možné úrovně pohybu (viz Ilustrace 5.8). Na těchto úrovních jsou s pomocí aplikace pro design úrovní, kterou vytvořil další člen týmu, rozmístěny jednotlivé objekty.



Ilustrace 5.8: Zobrazení devíti kolejí, po kterých se hrdina hry může pohybovat

Při hře se nanorobot neustále pohybuje vpřed, není možné zpomalovat ani zrychlovat. Hráč musí dávat pozor, které objekty se k němu blíží a vyhýbat se předmětům, které jsou lidskému krevnímu řečišti přirozené při sbírání předmětů pro vyléčení nemoci (například kapky tuku a tukové buňky pro vyčištění žíly při infarktu – na Ilustraci 5.8).

Aby se nanorobot mohl lidským tělem pohybovat, potřebuje nějaký pohon. Za ten byl zvolen cukr (jedna ze sbíraných surovin) jako klasická látka, kterou je možné v krevním řečišti najít. Hráč je motivován chovat se v řečišti co nejšetrněji, a tak se musí vyhýbat dalším pro člověka potřebným objektům (vitamíny, krvinky, atd.). I cukru musí sbírat jen co nejmenší množství, jaké potřebuje. Jinak se mu snižuje herní skóre. Vzhledem k tomu, že se cukr neustále snižuje, není tento úkol zcela triviální.

Pro dokončení jednotlivých úrovní je nutné splnit různé podmínky spočívající v nasbírání určitého počtu předmětů různých typů. Samozřejmě je také nutné dojet až na konec úrovně. Délka úrovní se s postupem ve hře zvětšuje.

5.2.2 Ovládání

Pohyb nanorobota mezi jednotlivými kolejemi je možné ovládat jak na klávesnici, tak na gamepadu. XNA umožňuje snadnou podporu obou těchto způsobů. Pro ovládání jsem vytvořila třídu `InputManager`.

Po testování mnoha různých nastavení se ovládání ustálilo na dvou typech pro klávesnici a jednom typu na gamepadu. Tam lze pro pohyb robota využít levé analogové páčky. Toto ovládání je typické pro mnoho současných her. Na klávesnici se hra může ovládat s pomocí šipek či s pomocí kombinace kláves „S“, „X“ a „K“, „M“ pro oba směry pohybu v osách X a Y.

Třída `InputManager` obsahuje funkce, které kontrolují stisknutí významově stejných kombinací pro různé ovladače. Například při testu, zda byla při hře stisknuta klávesa „nahoru“ se v této funkci testuje pohyb analogové páčky u gamepadu, stisk klávesy K a šipky nahoru.

Při každé aktualizaci herního stavu se načte současný stav kláves na klávesnici a tlačítek na gamepadu. `InputManager` takto uchovává vždy současný a předchozí stav. Ty se poté využívají pro detekci stisku ve zmíněných funkcích. Tyto funkce využívají jednotlivé herní stavy ve svých aktualizačních funkcích či například třída herního hrdiny `Hero`.

5.2.3 Implementace herní logiky

Třídy používané pro implementaci herní logiky byly naznačené v ilustraci 5.7. Ve třídě `Hero` se nachází veškeré proměnné potřebné pro zobrazování a aktualizaci stavu hlavního hrdiny nanorobota. Její součástí je jeho model, efekt (shader) používaný pro jeho vykreslování, proměnné potřebné pro pohyb (maximální akcelerace, zda-li se právě nachází mezi kolejemi a podobně) a v neposlední řadě i informace o množství zbývajícího pohonu. Při aktualizaci se toto množství po daném časovém intervalu snižuje.

Během tvorby projektu bylo implementováno několik typů kamer (například zcela nehybná a na pružině). Nakonec však byla vybrána kamera, která se pohybuje pouze v ose X dle pozice hrdiny. Příliš pohybující se kamera totiž způsobovala větší nepřehlednost ve hře.

Asi nejdůležitější třídou zpracovávající herní logiku je třída `Stage`. Ta načítá, vykresluje a aktualizuje herní úroveň. Ze souboru se sice úroveň načítá do třídy `StageData` (více v kapitole 5.3), nicméně textury, model a podobně se načítají až při vytvoření nové instance třídy `Stage`. Při tom se jednotlivé objekty vloží do několika seznamů podle typu a přes tyto seznamy se také vykreslují a aktualizují.

Existují dva typy objektů – objekty prostředí a objekty kolejí. První typ jsou objekty, s nimiž hráč nemůže nijak interagovat. Jedná se například o částicové efekty, ale i o různé ozdoby pro dokreslení živého světa lidského těla. Tyto objekty jsou všechny vloženy do jednoho seznamu (`environmentList`).

Dalším typem jsou objekty, které se vyskytují na kolejích a hrdina s nimi interaguje. Tyto objekty jsou uloženy do seznamů podle koleje a počítají se s nimi kolize. Tyto objekty se dále dělí na animované a neanimované, nicméně oba typy mají společnou bázovou třídu `InteractiveItem`, a tak je s nimi možné pracovat stejným způsobem. Jejich rozdíly se projevují pouze při aktualizaci a renderování obsaženého billboardu (výpočet animace).

Pro zpestření herního prostředí všechny objekty v krevním řečišti mírně pulzují. Tohoto efektu je dosaženo s pomocí matematické funkce sinus. Aby objekty pulzovaly odlišně, je jim při vytváření přiřazen náhodný offset. Ten se vždy přičítá k uplynutému časovému intervalu, který se předává jako parametr funkci sinus. Výsledek funkce sinus pak udává odchylku od počáteční pozice objektu.

5.2.4 Kolize a reprezentace objektů ve hře

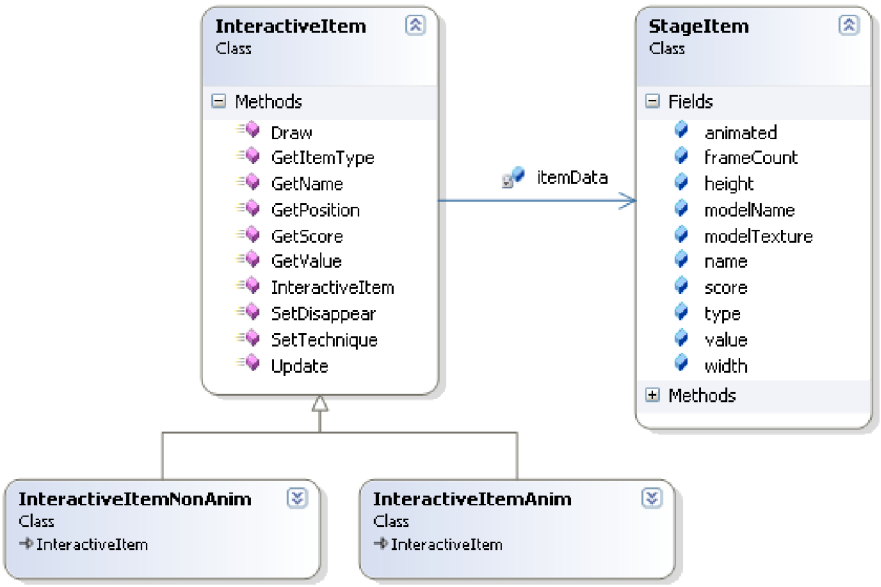
Výpočet kolizí se provádí také ve třídě `Stage`. Vzhledem k tomu, že jsou všechny interaktivní objekty pouze 2D, není nutné provádět složitý výpočet průsečíku obalových těles. Stačí pouze zjistit, zda-li je jejich pozice dostatečně blízko k pozici hlavního hrdiny. Pokud ano, nastává kolize.

Výpočet kolize se liší, pokud je hrdina na některé z kolejí, či pokud právě mezi kolejemi přejíždí. V prvním případě pak stačí projít pouze seznam dané koleje a zkontrolovat blízkost souřadnice objektů v ose Z. Pokud se hrdina nenachází na žádné koleji, je třeba zkontrolovat všechny objekty a blízkost všech tří souřadnic.

Objekty se dělí do několika kategorií a podle nich se pak mění reakce programu na kolizi. Tyto kategorie jsou:

- `Obstruction`,
- `Friend`,
- `Enemy`,
- `Particle`,
- `Neutral`.

Kategorie `Particle` a `Neutral` nemusíme v případě kolizí uvažovat, protože se nacházejí pouze mimo koleje a ve zvláštním seznamu (`environmentItems`). S objekty ostatních typů kolize nastat mohou. `Obstruction` označuje všechny objekty, do kterých by hrdina neměl vrazit (tělu prospěšné krvinky, vitamíny a podobně). V případě nárazu se ozve zvuk pro neúspěch a hráči se odečte skóre podle předmětu, s nímž nastala kolize. Každý objekt má totiž nejen informace potřebné pro své vykreslení (textura, výška, šířka, zda-li je animovaný), ale i informace o svém typu (`type`), hodnotě (`value`) a hodnotě pro skóre (`score`). Přehled těchto tříd můžete vidět na obrázku 5.9.



Ilustrace 5.9: Třídy reprezentující objekt ve hře

Hodnota `value` udává množství, o které se sníží či zvýší příslušný ukazatel. V případě `Obstruction` se jedná o ukazatel zdraví pacienta. Hodnota pro skóre se pak přičítá k hodnotě celkového skóre v rámci herní úrovně (tato hodnota může být i záporná). To se udržuje s pomocí třídy `StageScore`.

Jak je možné vidět na diagramu třídy StageItem, tak herní objekty byly navrženy co nejrobustněji. Například položka `modelName` je pro pozdější možné použití dalších 3D objektů (v současné verzi hry je 3D pouze model herní úrovně a hrdina).

Kategorie `Friends` (cukry, stopy do centra nemocí) a `Enemy` (buňky léčených nemocí – malarie, AIDS) se chovají podobně. Také přičítají svoji hodnotu k příslušným ukazatelům a ke skóre. Původním záměrem bylo různorodější chování těchto kategorií, nicméně vzhledem k tomu, aby hra byla pro hráče srozumitelná bylo od těchto nápadů upuštěno.

5.2.5 Zvýšení přehlednosti

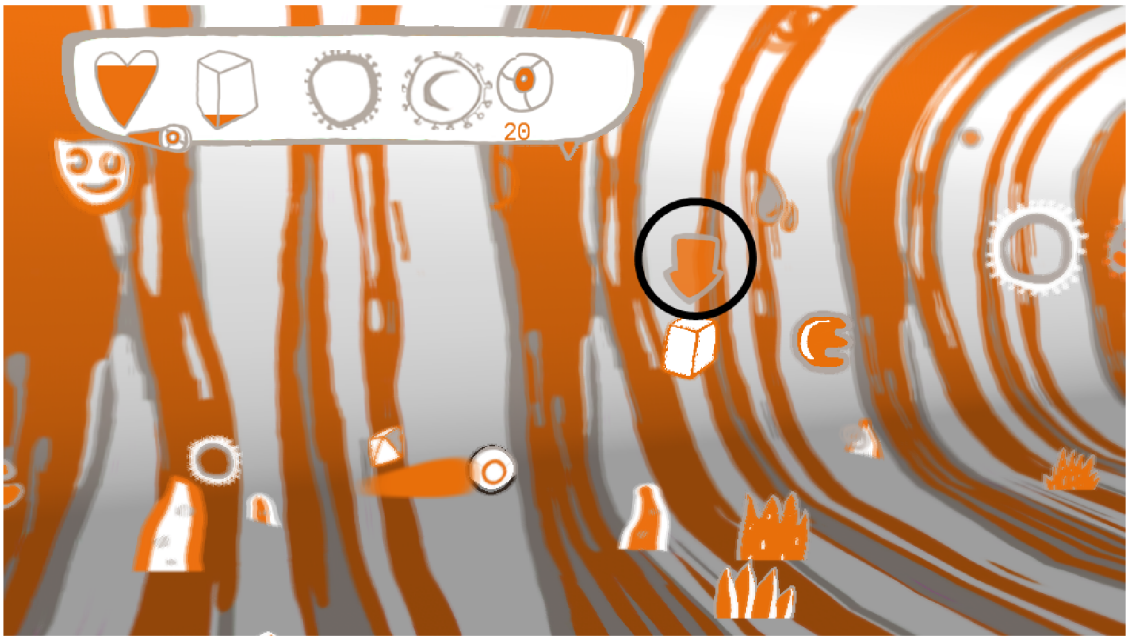
Hře NanoHeal byla již od počátku vývoje vytýkána špatná přehlednost ve hře a složité rozlišování mezi jednotlivými úrovněmi hloubky v ose X. V závislosti na tuto zpětnou vazbu bylo učiněno několik opatření. Prvním je výraznější zmenšení objektů na zadních kolejích (v ose X). Perspektivní zmenšení se totiž ukázalo jako nedostatečné. Předměty na kolejích 1,4 a 7 mají proto 80% své původní velikosti. Na nejvzdálenějších kolejích 0,3 a 6 pak pouze 60%.

Dalším opatřením byla implementace upraveného efektu Depth of Field. Ten zvýrazňuje aktuální hloubku, na které se nachází hrdina a rozmazává objekty podle vzdálenosti, v jaké se nacházejí od jeho pozice. Vzhledem k rozdělení této práce jsem podrobnější popis Depth of Field efektu zařadila do kapitoly věnované grafickým efektům (kapitola 6).

Posledním opatřením byla implementace zaměřování (target). Třída `Target` vykresluje šipku nad nejbližším interaktivním objektem, který se nachází ve stejné hloubce jako hrdina (viz Ilustrace 5.10). Přeškrtnutou šipku pak vykresluje v případě, že by se hráč měl tomuto objektu vyhnout. Pro

správný výpočet pozice šipky je třeba provést perspektivní projekci pozice nejbližšího objektu. Tento objekt je mu předán jako parametr aktualizační funkce. Musí se však vzít v potaz i její velikost vzhledem ke zmenšení objektů v zadních kolejích. Ta se upravuje podle souřadnice Z po perspektivní projekci (tedy podle hloubky).

Mezi dalšími uvažovanými opatřeními byla například úprava zorného pole kamery (FOV). Nicméně se ukázalo, že změnou tohoto nastavení se zlepšení v přehlednosti projeví pouze minimálně. Efekt mlhy, který se v současné době ve hrách pro zdůraznění hloubky hojně využívá, byl pak odmítnut kvůli grafickému konceptu tří barev, do nějž by nezapadal.



Ilustrace 5.10: Zvýraznění pomocné šipky při léčení nemoci AIDS

5.3 Práce s daty

Kromě multimediálních dat, která se zpracovávají přes řetězec zpracování herního obsahu (viz Kapitola 3.2.2) a dále za pomoci třídy XNA `ContentManager`, je třeba pracovat i s vlastními daty, ve kterých je uloženo nastavení programu či design herní úrovně. Vzhledem ke snadnosti použití a možnosti čitelnosti i mimo samotnou aplikaci byl pro ukládání těchto dat zvolen formát XML.

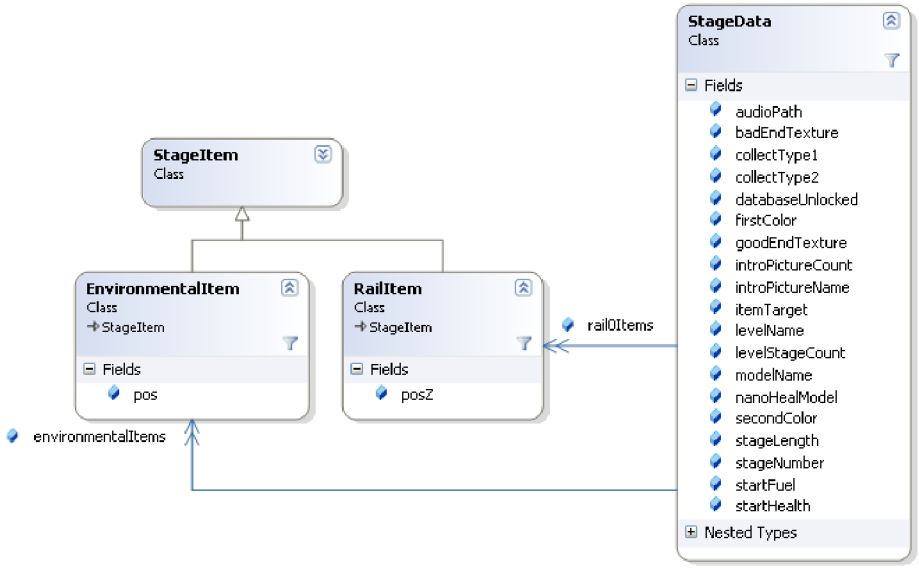
5.3.1 Načítání a ukládání herní úrovně

Data herní úrovně jsou uloženy ve třídě `StageData`. Jak lze vidět na ilustraci 5.11, tak tato třída obsahuje velké množství informací (část položek musela být vzhledem k rozsáhlosti skryta). Jednou z jejích klíčových funkcí je ukládání rozložení objektů v herní úrovni. Pro interaktivní objekty umístěné na kolejích využívá seznamy třídy `RailItem`. Vzhledem k tomu, že tyto objekty jsou uloženy v seznamech podle koleje, na které se nacházejí, je pro ně ukládána pouze pozice v ose Z. Při načítání se pak podle identifikace koleje dopočítají další souřadnice.

Pro objekty, které se mohou nacházet kdekoliv v herní úrovni je pak vytvořena třída `EnvironmentalItem`. Pro tu se v souboru ukládají všechny tři souřadnice. Společné prvky pak objekty mají specifikovány ve třídě `StageItem` (podrobněji se implementací této třídy zabývá předchozí podkapitola 5.2.4).

Dalšími daty, která je nutno vzhledem k barevné odlišenosti jednotlivých úrovní a možné rozšiřitelnosti o další odlišené úrovně ukládat, jsou textury pro herní menu (ukazatele sbíraných objektů, barevně odlišený ukazatel zdraví pacienta a cukrů) a další textury potřebné pro sladěné grafické ladění ve hře (šipka pro posun v animaci, logo u nahrávací obrazovky a podobně).

Pro správné odblokovávání následující úrovně musí data pro herní úroveň obsahovat pořadí úrovně v rámci jedné nemoci a celkový počet úrovní v ní. K zajištění variability úrovní jsou pak ukládána počáteční nastavení surovin a samozřejmě množství, které musí hráč nasbírat pro vítězství v úrovni. Díky nastavení jejich typu v rámci aplikace pro design úrovní je také možné snadno měnit sbírané objekty a vytvářet nové a graficky odlišné úrovně.

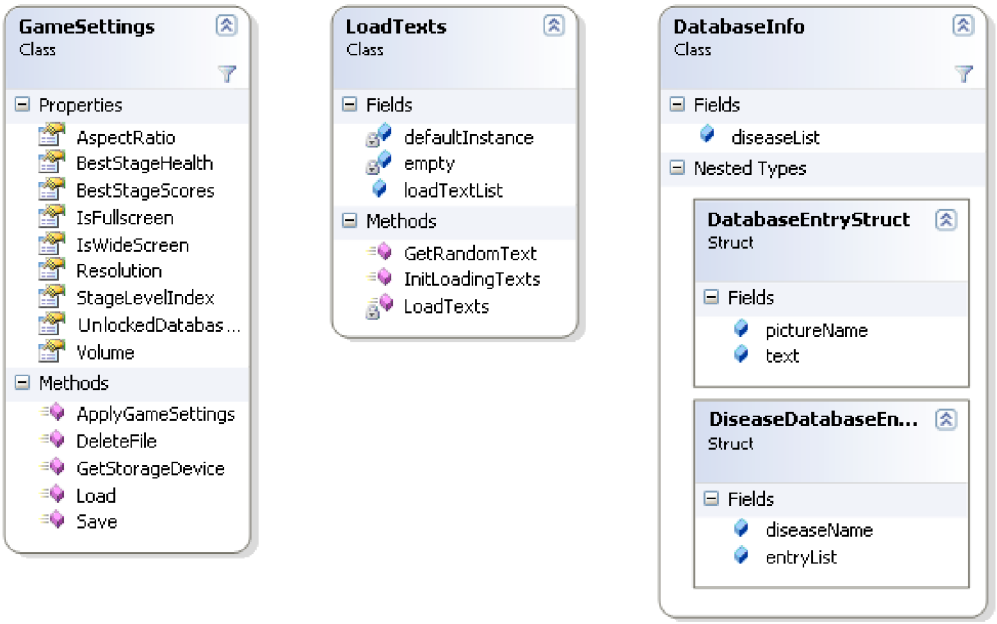


Ilustrace 5.11: Třídy pro uložení dat herní úrovně v externím souboru

Data se po výběru úrovně načítají s pomocí deserializace z XML dokumentů. Ty jsou pojmenovány podle léčené nemoci s indexem, který udává pořadí úrovně (např. Malaria0, Infarct1,...). Tento přístup umožňuje vytvářet propracované úrovně a jednoduchým způsobem je zakomponovat přímo do hry. Pro ukládání a načítání informací ze souboru je vytvořena třída `StageLoader`.

5.3.2 Další externí data

Podobný princip je implementován i při načítání a ukládání herních nastavení, vstupů do databáze a poučných textů při nahrávání hry. U nastavení se ukládá nastavení herního rozlišení, stav přepnutí na celou obrazovku, informace o skóre či zdraví v již odblokovaných úrovních nebo také počet zpřístupněných vstupů v databázi. Veškeré funkce a data pro práci s nastavením je možné najít ve třídě `GameSettings` (viz Ilustrace 5.12).



Ilustrace 5.12: Rozhraní třídy pro ukládání herního nastavení a další třídy pro ukládání externích dat

Třída pro herní nastavení obsahuje svoji privátní statickou instanci a pouze privátní konstruktor. Je tak zajištěn vznik pouze jedné instance této třídy. K jednotlivým nastavením se pak přistupuje pouze funkcemi rozhraní třídy. Po spuštění aplikace se nejprve načte (stejně jako v případě herní úrovně s pomocí deserializace) předem uložené nastavení. Pokud není nalezeno, použije se výchozí nastavení, které je dodáváno se hrou. Při každé změně v grafickém nastavení či při odblokování další úrovně se pak toto nastavení uloží i do souboru.

Stejným způsobem (jedna instance) funguje i třída pro výběr zobrazených informací o jednotlivých nemocích při nahrávání herní úrovně (`LoadTexts`). Je třeba ji inicializovat před požadavkem na náhodný text ze souboru. Při inicializaci se z něj načte seznam řetězcových proměnných, proměnná `empty` se nastaví na `false` a je možné použít funkci `GetRandomText()`.

Posledním typem externě uložených dat je obsah databáze. Pro každý vstup je uložena řetězcová proměnná s jeho názvem. Ten se využívá při vytváření menu na obrazovce databáze (viz Ilustrace 5.13). Samotný informační text je pak uložen jako obrázek ve formátu .png.

Ilustrace 5.13: Obrazovka herní databáze hry NanoHeal

Práce s externími daty (především zápis do nich) se ukázala být jediným výraznějším problémem při převodu hry pro konzolu Xbox 360. Proto se touto problematikou budu zabývat ještě v podkapitole 5.5.2.

5.4 Hudba a zvuky

Nezbytnou součástí každé hry je v dnešní době stejně jako u filmu hudba a zvuky. Platforma XNA umožňuje několik způsobů jejich přehrávání. V projektu NanoHeal bylo využito nástroje XACT, o němž se zmíním v následující podkapitole. Nicméně v případě konverze projektu pro přehrávač Zune by bylo třeba využít jednoduché vestavěné třídy pro přehrávání zvuku – `SoundEffect`. XACT není přehrávačem Zune podporován [2].

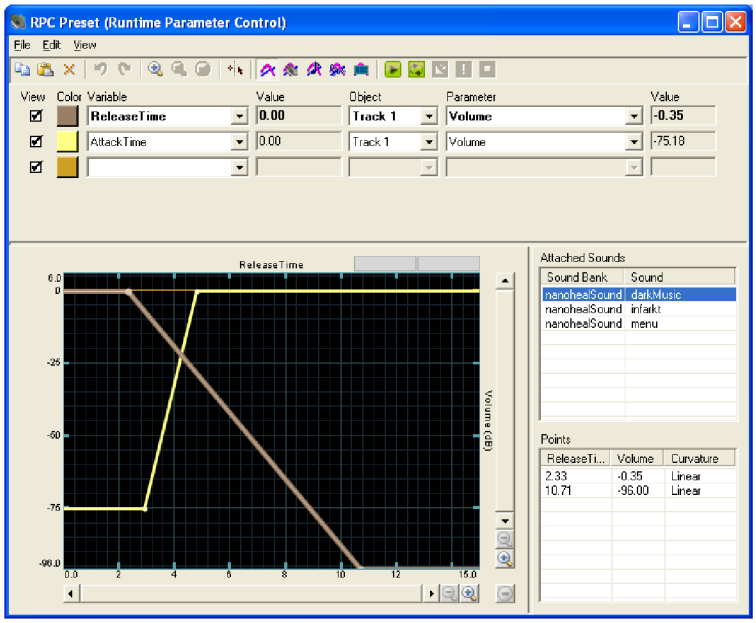
Tímto způsobem lze jednoduše přehrát zvukové formáty, které jsou do projektu importovány přes řetězec zpracování herního obsahu (content pipeline). Mohou jimi být například soubory ve formátu .mp3, .wav či .wma.

5.4.1 XACT

XACT neboli Microsoft Cross-Platform Audio Creation Tool je srdcem aplikačního rozhraní XNA pro audio. Kromě tříd a funkcí, díky kterým můžeme vytvořené zvuky přehrávat obsahuje i aplikaci, která umožňuje sdružovat zvuky do skupin, komprimovat je, vytvářet různé přechody mezi zvuky a také definovat proměnné, které ovlivňují chování zvuků přímo ve hře [2]. S jeho použitím máte také možnost přehrávání zvuků ve 3D prostoru. Obsahuje i monitorování přehrávání zvuků pro ladící účely (Audition Monitor).

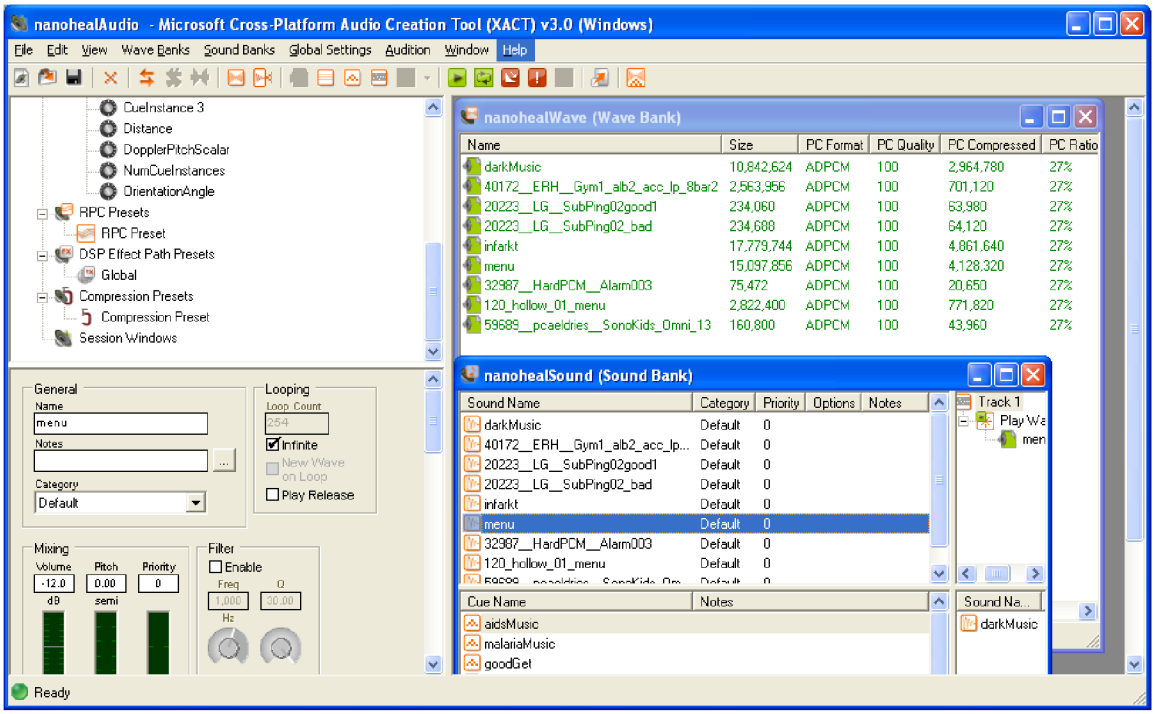
Na ilustraci 5.14 můžeme vidět nastavování přechodu mezi zvuky za pomocí křivek v grafu. Žlutá křivka symbolizuje zvyšování hlasitosti při nástupu skladby. Hnědá pak snižování při ukončení

jejího přehrávání. S pomocí křivek můžeme kromě hlasitosti (volume) nastavovat také výšku (pitch) a další efekty.



Ilustrace 5.14: Příprava přechodů mezi zvuky v XACT aplikaci

Pro přehrávání skrze aplikační rozhraní XACT je třeba zvuky vytvořit přes stejnojmennou aplikaci (viz Ilustrace 5.15). Jejím výstupem je soubor s nastaveními ve formátu .xap. Z něj a v něm definovaných zvukových souborů (podporuje pouze formát .wav) pak řetězec pro zpracování herního obsahu vytvoří soubory s wave a zvukovými bankami (soubory ve formátu .xwb a .xsb) a s globálním nastavením (formát .xgs).



Ilustrace 5.15: XACT aplikace

V horní části obrazovky vidíme wave banku. Ta reprezentuje všechny načtené zvuky formátu .wav. U těch je možné nastavovat například kompresi. Zvuky je možné sdružovat v jeden zvuk do zvukové banky (sound bank). Zde můžeme nastavit zda-li se mají přehrávat ve smyčce, změnit jejich hlasitost a podobně. Nakonec je třeba vytvořit jednotlivé stopy (cue), které lze přehrát v programu.

V projektu NanoHeal jsem vytvořila statickou třídu `AudioManager`. Ta umožňuje připravené zvuky z kteréhokoliv místa v programu přehrát. Pro práci se zvuky vytvořenými s pomocí XACT je třeba používat XNA třídu `AudioEngine`. Při jejím vytváření jí musíme předat soubor s globálním nastavením (.xgs soubor). Ke správné funkci musíme ale načíst i vytvořené wave (.xwb soubor) a zvukové banky (.xsb soubor).

K přehrávání hudby slouží statická funkce `PlayMusic()`, která uvolní případnou předchozí přehrávanou skladbu a začne hrát skladbu novou. Zvuky se přehrávají statickou funkcí `PlaySound()`. Poslední funkcí pro přehráváná zvuku je také statická `PlaySiren()`. Ta upozorňuje hráče na nedostatek paliva. Hromadné nastavování hlasitosti je možné s pomocí takzvaných kategorií. Výchozí kategorii všech zvuků ve hře NanoHeal využívá funkce `SetVolume()`.

5.5 Převod na Xbox 360

Hra NanoHeal byla již od začátku koncipovaná i pro konzolu Xboxu 360. Platforma XNA minimalizuje problémy při tomto převodu a osobně jsem se přesvědčila, že je tento proces rychlý a snadný.

Několik změn bylo nutné provést vzhledem k odlišnému HW vybavení (uvažování ovládání gamepadem) a odlišné koncepce systému (nemožnost vypnutí zobrazení na celou obrazovku). K těmto úpravám dobře posloužily příkazy pro podmíněný překlad. V XNA Game Studio je při překladu pro konzolu definováno makro `XBOX360`. Například nastavení pro přepnutí na celou obrazovku je v případě Xbox verze programu snadné vynechat s pomocí příkazu `#if !XBOX360`.

V XACTu (viz předchozí kapitola) je při tvorbě zvuků možné nastavit různé zvuky či různá nastavení zvuků pro Xbox a Windows verzi projektu. Při překladu se pak programátor nemusí nic dalšího ošetřovat. Jediná komplikace při převodu hry NanoHeal nastala při zápisu změny nastavení do externího souboru. O tomto problému se zmíním v podkapitole 5.5.2.

5.5.1 Spuštění

Pro spuštění hry na Xboxu je nejprve třeba nainstalovat aplikaci XNA Game Studio Connect na Xboxu 360 [21] a vytvořit verzi projektu pro Xbox. XNA Game studio umožňuje vytvoření konzolové verze projektu jednoduchým příkazem „Create Copy of Project for Xbox 360“. Po spuštění aplikace Game Studio Connect na Xboxu a její prvotní synchronizaci s XNA Game Studiem na PC je možné snadno a inkrementálně přenášet nové verze projektu na konzolu (příkaz deploy v Game Studiu). Tam se přenesená hra může spouštět stejně jako ostatní stažené aplikace přes knihovnu her – games library.

Jediným problémem při přenosu bylo nestandardní pojmenování jedné z testovacích textur (název obsahoval znak „+“). Při přenosu vznikla chyba a z chybového hlášení nebylo zcela patrné, že chyba je v pojmenování jednoho ze souborů.

5.5.2 Práce s daty

S daty uloženými ve formátu XML (podrobněji jsou externě ukládaná data popsána v kapitole 5.3) je možné pracovat stejným mechanismem (serializace a deserializace) i na konzole. Problém nastává při hledání lokace, která je platná pro zápis dat na Xboxu i na PC. Zpočátku vývoje jsem k ukládání dat využívala kořenový adresář herního obsahu (tedy stejná lokace jako lokace .exe souboru pro spuštění hry). Tento přístup ale na konzole, která může mít několik pevných disků či paměťových karet, nelze použít (navíc v tomto adresáři aplikace na konzole nemá práva k zápisu). K výběru místa uložení tak, aby tento mechanizmus fungoval korektně na PC i Xboxu pomáhá třída `StorageDevice` [22].

K její správné inicializaci je třeba vyzvat hráče pro výběr úložiště skrze dialogové okno označované jako Guide skrze stejnojmennou třídu. V případě spuštění na PC se data budou ukládat automaticky do adresáře `Documenty/SavedGames`. Pro korektní fungování dialogu `Guide` je třeba do programu přidat komponentu `GamerServicesComponent`. Tato komponenta kromě služby dialogu `Guide` nabízí i další služby spojené s hraním ve více hráčích přes internet.

Výsledkem dialogu `Guide` je objekt třídy `StorageDevice`, která obsahuje korektní lokaci pro zápis ukládaných dat. Dialog je lepší vyvolávat asynchronně proto, aby nezastavil další provádění programu.

5.6 Ladění a distribuce

Vzhledem k rozsáhlosti projektu bylo nutné věnovat zvláštní úsilí ladění. Hojně jsem využívala podmíněného překladu a speciálních ladících výpisů v případě ladící verze programu (makro `DEBUG`). Při ladění grafických efektů jsem využila pomocnou funkci vykreslení textury na obrazovku a jednotlivé fáze efektů (při nichž bylo třeba obraz vykreslit víckrát) jsem mohla jednoduše vykreslovat podle stisknuté klávesy.

Velmi se také osvědčila úprava herní logiky pro účely ladění. Hrdina se v tomto módu může pohybovat i dozadu a může zrychlovat, zpomalovat a v žádném případě nedojde ke konci hry před koncem herní úrovně (například kvůli nedostatku pohonu). Také je možný libovolný pohyb kamerou s pomocí znakových kláves.

Ladění na Xboxu probíhá téměř totožně jako na PC. Je možné používat body přerušení programu (break pointy) a podobně. Speciálně pro ladění na Xboxu XNA nabízí ještě aplikaci XNA Framework Remote Performance Monitor. Ta umožňuje sledovat výkon a například ukládat obsah paměti v jistém okamžiku aplikace. Tato aplikace se velmi osvědčila především ke konci projektu při problémech se špatným uvolňováním herního obsahu při přechodech mezi jednotlivými úrovněmi.

Pro distribuci projektu je možné využít několika přístupů. Pro programátora je nejsnadnější variantou využít Game Studiem přímo podporované zabalení do formátu .ccgame (verze pro PC a konzolu je třeba zabalit zvlášť). Tento formát pak mohou jak další vývojáři s nainstalovaným XNA Game Studiem bez problémů rozbalit a hru spustit. Pokud chceme hru rozšířit i běžným uživatelům na PC (na Xboxu je již pak jen možnost placeného účtu a distribuce přes online obchod), tak je

možné vytvořit vlastní instalátor s knihovnami potřebnými pro běh programu (XNA Framework Redistributable).

V tomto případě je třeba zkontrolovat kompatibilitu se všemi použitými komponentami, které v tomto balíku nemusí být zahrnuty. Například komponenta `GamerServicesComponent` používaná pro nalezení úložiště na Xboxu v něm není. Ve verzi pro co nejmenší distribuci na PC jsem tedy musela práci s touto komponentou vynechat a vrátit se k původnímu způsobu ukládání dat.

Ke tvorbě instalátoru jsem využila před-připravený instalátor s pomocí WiX (Windows Installer XML) [23]. WiX je soubor nástrojů pro tvorbu instalátorů pro operační systém Windows. Tento instalátor zkontroluje u uživatele přítomnost právě knihoven XNA a .Net Frameworku, který je taky nutný pro běh aplikací v XNA. Zkontroluje také potřebnou podporu u grafických karet (shader model).

Výsledná velikost instalátoru NanoHeal.msi, který obsahuje knihovny XNA je 35,1MB. Vzhledem k tomu, že instalátor samotného XNA Game Studia, který by bylo třeba nainstalovat v případě distribuce .ccgame, je přes 72MB, vykazuje využití WiX velkou úsporu a zvýšení pohodlí uživatelů.

6 Grafické efekty

V této kapitole bych ráda zmínila implementované grafické efekty a techniky, které byly použity pro co nejpřitažlivější výsledný vzhled hry NanoHeal. Nejprve bude popsán postup k dosažení nerealistického vzhledu hry, pak renderování 2D prvků, které bylo třeba nerušivě vložit do 3D prostředí. Dále se budu zabývat post-processingem scény (zpracováním vykreslené scény), kde byl využit upravený efekt „Depth of Field“. Poslední podkapitola bude věnovaná částicovým systémům, bez kterých je v dnešní době grafika hry téměř nepředstavitelná.

6.1 Nerealistické vykreslování

Jak bylo zmíněno v kapitole 3.4, tak si tento projekt klade za cíl využít technik nerealistické grafiky k dosažení co nejlepšího grafického vzhledu. Těchto technik je dosaženo především s pomocí shaderů, tedy efektů.

6.1.1 Zobrazování hrdiny ve stylu cartoon

Model hlavního hrdiny hry, který symbolizuje nanobota, je renderován víceprůchodově. Aby mohla být zjištěna přesná silueta modelu, je nejprve nutné získat informace o normálách modelu a také informace o hloubce.

Metody pro vykreslování hrdiny stejně jako aktualizaci jeho stavu a další funkce, které se ho týkají obsahuje třída `Hero`. Shadery použité pro jeho renderování se pak nachází souboru Hero.fx. V tomto souboru jsou klíčové především dvě techniky (dvoje nastavení kombinace shaderů pro renderování).

Jednou je `DepthNormal`, jejímž výstupem je normalizovaná normála převedená do světových souřadnic. Tato normála je zapsaná jako výsledná barva do nastaveného cíle vykreslování (angl. render target). Vzhledem k tomu, že výsledkem pixel shaderu je vektor o čtyřech prvcích a normála zabírá pouze první tři, je poslední prvek využit k přenesení informace o pozici vertexu v ose Z po perspektivní projekci, tedy o hloubce v obrazu. Cílem vykreslování je v tomto případě textura o stejné velikosti jako je rozlišení obrazovky. Tato textura se pak předává následující technice jako parametr.

Druhou technikou je `Outline`. Ta vykreslí nanorobota i s linkou zvýrazňující siluetu a s jednoduchým kvantifikovaným stínováním. Nejprve se vypočítá souřadnice do textury s informacemi o normále a hloubce z pozice pixelu na obrazovce. Následně se získají čtyři vektory s informacemi o sousedních normálách – uvažuji čtyři rohové sousedy zpracovávaného pixelu. Z těchto vektorů je pak možné vypočítat změnu D v diagonálách (viz (1)). Dvojce vektorů v_1, v_2 a v_3, v_4 jsou vždy protilehlými vektory.

$$D = |\vec{v}_1 - \vec{v}_2| + |\vec{v}_3 - \vec{v}_4| \quad (1)$$

V závislosti na míře velikosti změny se pak určí, zda-li pixel náleží do siluety modelu a měl by se tedy vykreslit barvou obrysu či ne. Míra velikosti změny nám poslouží jako faktor pro míchání s pomocí funkce pro výpočet lineární interpolace. Interpoluje se mezi požadovanou barvou obrysu

(která není vždy černá) a současnou barvou pixelu. Pro dosažení lepších výsledků jsou velmi malé změny v hloubce i v normále zanedbávány.

V technice `Outline` se kromě výpočtu a vykreslení obrysu modelu počítá i stínování modelu. Při výpočtu míry osvětlení využijeme vzorce, který vychází z klasického Lambertova osvětlovacího modelu pro výpočet difúzního světla (2). Míra osvětlení (cosinus vektorů N a L) *Light* se získá skalárním součinem normalizované normály N a normalizovaného směru osvícení L (uvažuje se směrový typ světla).

$$Light = \vec{N} * \vec{L} \tag{2}$$

K dosažení cartoon vzhledu (viz Kapitola 3.4.1) jsou implementovány pouze dvě úrovně stínování – se stínem a bez stínu. V případě, že je tedy míra osvětlení menší než daná hranice, vykreslí se barva stínu (barva textury snížená o konstantní hodnotu).

6.2 Renderování 2D prvků

Vzhledem ke grafickým požadavkům vzneseným na hru (viz Kapitola 4) bylo nutné najít způsob, jak vhodně zkombinovat modelovanou 3D grafiku s 2D prvky prostředí. Jako řešení byly zvoleny tzv. billboardy, které se ve hrách často používají pro nahrazení 3D prvků 2D prvky. Tyto 2D prvky jsou neustále rotovány tak, aby směřovaly ke kameře a vytváří tedy iluzi, že se jedná o 3D prvky. Často se tato technika využívá především pro rostliny (např. tráva, stromy v dálce) či u částicových efektů (např. kouř, oheň).

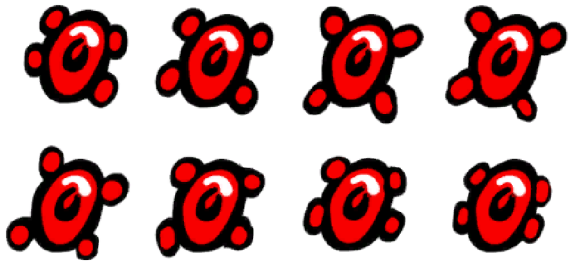
Billboardy je možné v tomto projektu využít především proto, že je ve scéně pouze fixní kamera. Při kameře ovládané hráčem by byl efekt při některých úhlech a pohybech patrný.

6.2.1 Úvod do použitých billboardů

Ve hře NanoHeal jsou billboardy implementované kompletně na GPU, tedy v shaderech. V programu jsou reprezentovány čtyřmi vrcholy se stejnou pozicí, ale odlišnými texturovými souřadnicemi. Z těchto informací se pak v shaderech vytvoří obdélník natočený na kameru. Textura billboardu obsahuje alfa kanál. Při použití alfa blendingu tedy není patrné, že prvek herního prostředí je vykreslený na obdélníku.

Aby prostředí působilo živě a bohatě, byl vznesen požadavek (viz Kapitola 4), že by tyto billboardy měly být animované. Jedna textura billboardu se tak může skládat z jednotlivých snímků animace (viz Ilustrace 6.1). Snímky animace se mění v závislosti na čase. Shaderu se předá informace o požadovaném snímku a ten pak správně vypočítá texturové souřadnice. Funkce pixel a vertex shaderu pro vykreslení animovaného billboardu jsou v souboru AnimatedBillboard.fx pro animované billboardy a v souboru Billboard.fx pro neanimované. Programové třídy, které se starají o vykreslování a případně animování billboardů se jmenují také `AnimatedBillboard` a `Billboard`.

Pro zvýšení efektivity se vykreslují pouze některé billboardy. Rozhodnutí o vykreslení závisí na vzdálenosti od hlavního hrdiny. Pokud billboard nemůže být hráčem viděn, nevykreslí se. V případě, že nastane s billboardem kolize, nastaví se příznak `disappear` v příslušných třídách na `false` a do pixel shaderu se předává stále se zvyšující hodnota, o kterou se snižuje alfa hodnota výsledné barvy. Billboardy tak plynule zmizí.



Ilustrace 6.1: Ukázka animované textury

6.2.2 Řešení aplha blendingu

Vykreslení billboardů probíhá kvůli korektnímu míchání barev (angl. blending) ve dvou průchodech. Vzhledem k plánovanému vysokému počtu billboardů ve scéně, by ruční třídění podle hloubky bylo příliš výpočetně náročné. Proto se při prvním průchodu vykreslí pouze neprůhledné části textur bez alfa blendingu a se zapnutým hloubkovým bufferem. Tím zajistíme jejich správné setřídění a vykreslení.

Při druhém průchodu se buffer hloubky nastaví pouze pro čtení, zapne se alfa blending a vykreslí se průhledné části. I přesto, že se tímto neúplným setříděním dopouštíme chyby, na výsledném obraze to není poznat a výpočetní úspora výkonu proti ručnímu setřiďování je velká.

6.2.3 Shader billboardu

V aplikaci je billboard reprezentován čtyřmi vrcholy (vertexy) na jedné pozici – středem spodní části výsledného obdélníku. Jednotlivé vrcholy se liší pouze texturovými souřadnicemi – které jsou hraničními pro všechny rohy textury. V shaderu je třeba podle texturových souřadnic vypočítat správně jednotlivé body tak, aby byl obdélník natočený na kameru. K tomu slouží vektor *R* (označován angl. jako right), který je možné dopočítat vektorovým součinem pohledového vektoru a normálového vektoru (3). Výsledek je třeba normalizovat.

Pohledový vektor *View* (udává směr, ze kterého se na billboard dívá kamera) je možné získat z pohledové matice kamery. Jedná se totiž o její prvky. *N* udává normálu billboardu. V tomto případě je normála pevně nastavená směrem nahoru v ose Y.

$$\vec{R} = \vec{View} \times \vec{N} \tag{3}$$

Výsledná pozice se pak vypočítá podle vzorců (4) a (5). Vektor *pos* obsahuje pozici vrcholu, *texCoordX* a *texCoordY* udávají texturové souřadnice v osách X a Y a *width* a *height* udává výslednou šířku a výšku billboardu. Texturové souřadnice je pak ještě nutné v případě animovaného billboardu přepočítat tak, aby ukazovaly na korektní snímek animace.

$$\vec{pos} = \vec{pos} + \vec{R} * (texCoordX - 0.5) * width \tag{4}$$

$$\vec{pos} = \vec{pos} + \vec{N} * (1 - texCoordY) * height \tag{5}$$

Tento přístup billboard natáčí pouze vpravo a vlevo a při náletu nad billboardy by bylo možné pozorovat jejich plochost. Vzhledem k fixní kameře ale není třeba zajišťovat natáčení billboardů s pomocí druhého vektoru (nahoru/dolů).

6.3 Zdůraznění hloubky obrazu

K dodání hloubky hře a ulehčení orientace hráčů při přesuny mezi jednotlivými hloubkami bylo rozhodnuto o implementaci upraveného efektu označovaného jako Depth of Field (efekt zdůrazněné hloubky obrazu). Tento efekt je v nejnovějších hrách velmi populární a často využívaný. Jedná se o zaostření pouze na určitou hloubku v obraze a postupné rozostření bližších a vzdálenějších objektů (Ilustrace 6.2).

Nejprve bych ráda zmínila efekt Depth of Field (dále jako DoF) v jeho základní podobě a poté úpravy, které jsem provedla pro lepší výsledky ve hře NanoHeal. Stejně jako předchozí grafické efekty je i tento implementován s pomocí shaderů.

Ilustrace 6.2: Depth of Field efekt použitý ve hře LittleBigPlanet (zdroj [18])

6.3.1 Úvod do DoF

DoF se řadí mezi tzv. post-processingové efekty, které se aplikují na již vykreslenou scénu. Ta se obvykle renderuje do textury a při vykreslování této textury na obrazovku se na ni aplikuje pixel shader. DoF se skládá z několika kroků:

- vykreslení hloubky,
- vykreslení normální scény,
- rozmazání scény,
- sloučení rozmazané a nerozmazané scény podle hloubky.

Oba první kroky vyžadují vykreslení všech objektů scény do textury. Jedná se tedy o relativně výpočetně náročný efekt. Poté je nutné vykreslit texturu normální scény a aplikovat na ni shader rozmazání (anglicky blur). Dále se znovu vykreslí textura normální scény a v shaderu se smíchá s rozmazanou scénou podle textury s hloubkou.

Hloubkou používanou v klasické verzi efektu DoF je myšlena vzdálenost pixelu mezi rovinami „near“ a „far“ projekční matice. Po vynásobení bodu modelovou, pohledovou a projekční maticí získáme informaci o hloubce *depth* dle vzorce (6).

$$depth = \frac{\vec{pos}.Z}{\vec{pos}.W} \tag{6}$$

Při míchání rozmazané a normální scény s pomocí před-počítané hloubkové textury pak zadáme vzdálenost od kamery, která by měla být nejostřejší (ohnisko) a rozsah, který udává jak rychle se bude rozmazání se vzrůstající vzdáleností od ohniska zvyšovat. Podle vzorců (7) a (8) se vypočítá míchací index, který se použije ve funkci pro lineární interpolaci (viz [20]). Ta bude interpolovat právě mezi rozmazanou a normální scénou.

$$Dz = \frac{-Near * Far}{depth - Far} \tag{7}$$

$$index = \frac{|Dz - Distance|}{Range} \tag{8}$$

Near je hodnota blízké ořezávací roviny, *Far* je pak hodnota vzdálené ořezávací roviny převedené do intervalu [0,1]. *depth* je načtená hloubka z předem vypočítané hloubkové textury. *Distance* udává ohnisko a *Range* rozsah.

6.3.2 Implementace DoF

Pro hru NanoHeal implementace klasického způsobu rozostření podle hloubky po perspektivní projekci není vhodná kvůli nastavení kamery. Byla proto implementována upravená verze DoF založená na hloubce objektů pouze v ose X. Díky tomu bude hráč vidět nejostřeji všechny objekty, které se nacházejí ve stejné hloubce v ose X jako hrdina hry. Jeho pozice totiž udává vzdálenost ohniska.

Pro každý objekt renderovaný v 3D prostředí bylo tedy nutné doprogramovat shader pro výpočet hloubky *depth* dle vzorce (9). Vektor pos je pozice vrcholu vynásobená pouze modelovou maticí. *LevelWidth* je pak šířka modelu herní úrovně.

$$depth = \frac{\vec{pos}.X}{LevelWidth} \tag{9}$$

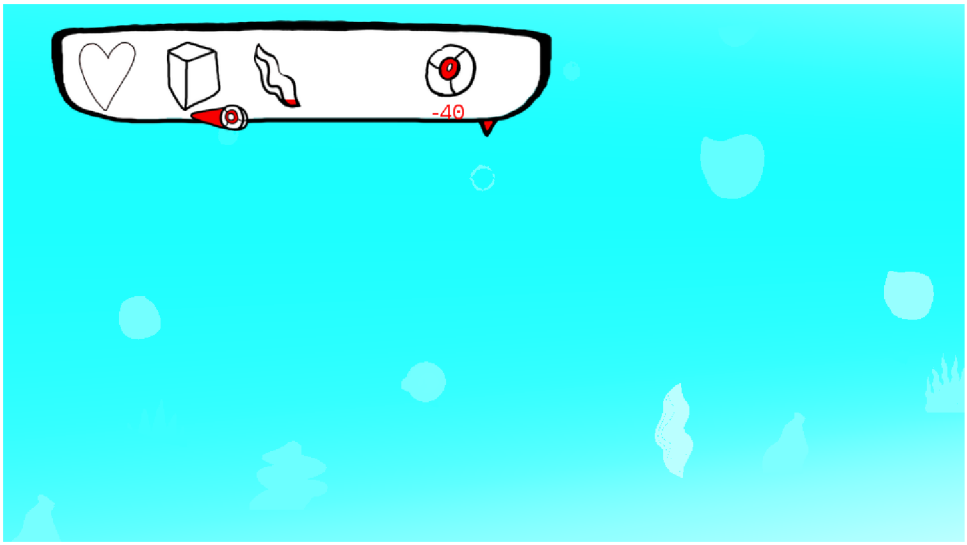
Index pro interpolaci rozmazané a normálně vykreslené scény se pak vypočítá dle (10). Proměnná *depth* je hloubka načtená z textury, *FocusPosX* udává pozici hrdiny v ose X a *Range* upravuje míru rozmazání. Proměnná *index* se stejně jako v klasickém způsobu použije jako index míchání při lineární interpolaci mezi rozmazanou a ostrou scénou.

$$index = \frac{|FocusPosX - (depth * LevelWidth)|}{LevelWidth} * Range \tag{10}$$

6.3.3 Vykreslování

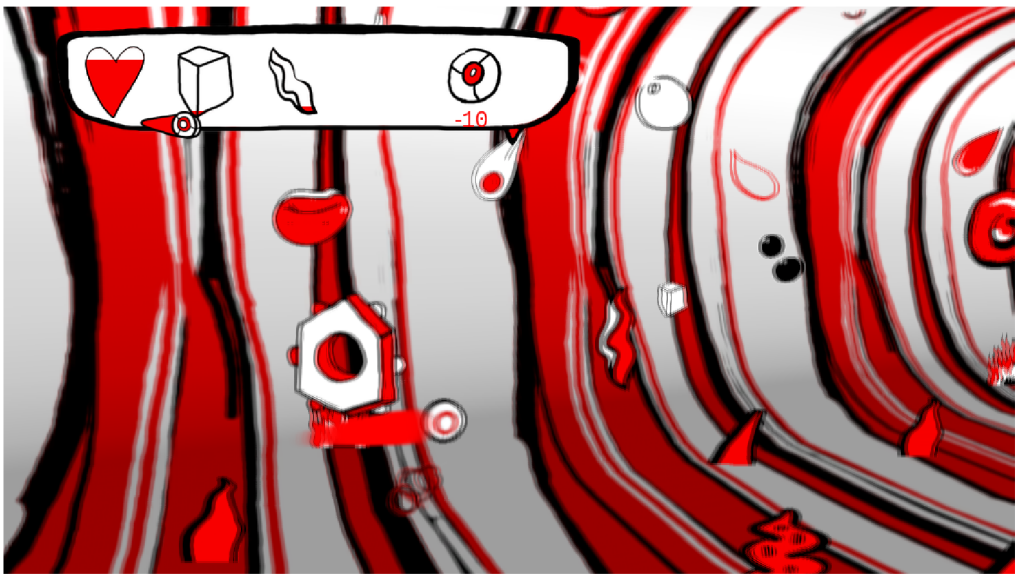
Renderování scény probíhá skrze funkci `RenderStage()` ve třídě `LevelManager`. Pro renderování scény do textury je třeba v XNA vytvořit nové cíle renderování (`RenderTarget`). Při

renderování scény a jejím rozmazávání si vystačíme s formátem totožným s hlavním cílem renderování (obrazovkou monitoru či televize). Nicméně při renderování hloubky je vhodné nastavit formát textury na `SurfaceFormat.Single`. Tento formát používá všech 32 bitů pouze pro červený kanál barvy. Když tedy hloubku, která se nachází v intervalu [0,1] uložíme do červeného kanálu, získáme větší přesnost. Ukázku hloubkové textury z projektu NanoHeal můžete vidět na Ilustraci 6.3.



Ilustrace 6.3: Hloubková mapa

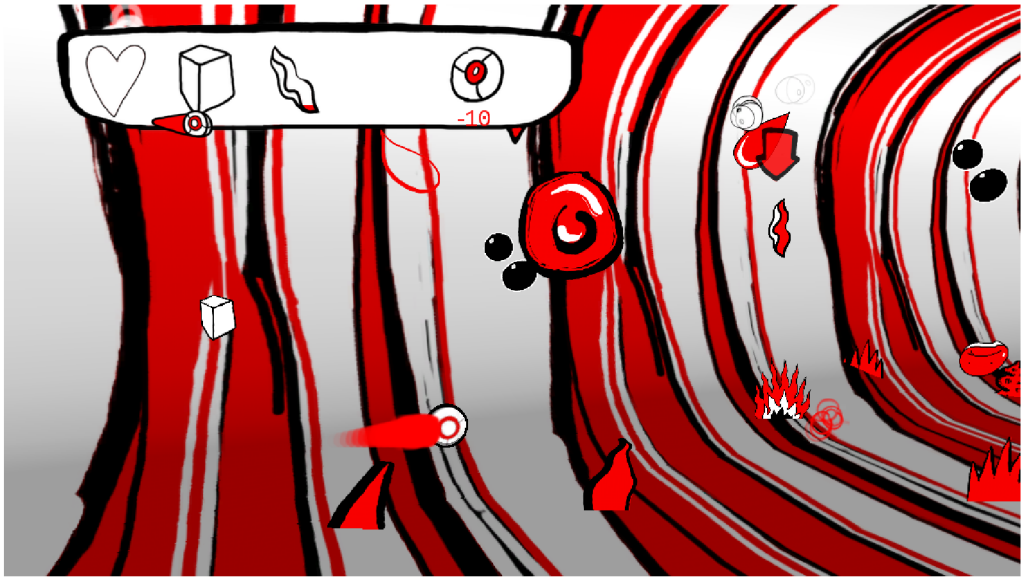
Nejprve se tedy postupně celá scéna (model herní úrovně, model hrdiny, částicové systémy a billboardy) vyrenderuje do textury se stejným nastavením jako hlavní cíl renderování. Ta je poté znovu vykreslena s použitím shaderu pro rozmazání do další textury (viz Ilustrace 6.4).



Ilustrace 6.4: Scéna s aplikovaným rozmazáním

Existuje mnoho druhů rozmazání (anglicky blur). Radiální rozmazání rozmazává obraz kruhově, Gaussův blur zase podle Gaussovy funkce. Pro účely projektu a pro co nejnižší zvýšení výpočetní náročnosti (protože rozmazání obrazu se často dělá pro vyšší kvalitu několika-průchodově)

bylo dostačující využít jednoduché průměrování. Pixel shader pro rozmazávání sečte informace o barvě ze čtyř rohových pixelů kolem právě zpracováváno a tuto informaci vydělí počtem použitých pixelů (tedy čtyřmi). Pro vyšší míru rozmazání nejsou použity přímo sousední rohové pixely, ale pixely s určitým offsetem. Vykreslení ostré scény můžete vidět na ilustrací 6.5.



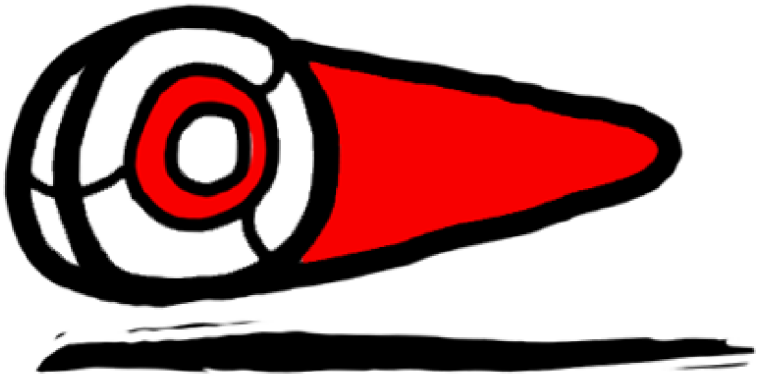
Ilustrace 6.5: Ostrá scéna

Nakonec zbývá jen předat shaderu pro sloučení rozmazané a ostré scény všechny potřebné textury a data a podle postupu v podkapitole 6.3.2 vy-renderovat a zobrazit finální scénu (Ilustrace 6.6).

Ilustrace 6.6: Scéna s DoF

6.4 Částicové systémy

Částicové systémy jsou v dnešní době nezbytnou součástí každé hry. Používají se pro simulaci různých přírodních efektů jakými jsou například kouř, déšť, či různých výbuchů, ale i vlasů a mnoho dalších jevů. Proto i v požadavcích na vývoj hry NanoHeal (viz 4.1.1) nechybí částicové systémy pro oživení herního prostředí. Vzhledem k zasazení hry do prostředí krevního řečiště bylo rozhodnuto o implementaci bublinek, které dokreslí atmosféru v kapalině. V neposlední řadě pak bylo třeba vytvořit s jejich pomocí také „vlající efekt“ za samotným nanorobotem, který byl v plánu hned od počátečních stádií návrhu grafiky (Ilustrace 6.4).



Ilustrace 6.7: Logo projektu NanoHeal

6.4.1 Použitá komponenta

Vzhledem k existenci kvalitní a rychlé implementaci nastavitelných částicových systémů na [1], jsem se rozhodla integrovat tento systém (pojmenovaný ParticleSample) do projektu NanoHeal. Tato komponenta se skládá z několika tříd. `ParticleSystem` je nejdůležitější z nich. Při vytváření vlastního částicového systému je nutné ji použít jako základní třídu pro odvozený systém. Její součástí je třída `ParticleSettings`. Ta obsahuje různé nastavení částic (velikost, barva, použitá textura, atd.). Dále obsahuje komponenta třídu `ParticleEmitter`. Ta má na starosti vytváření nových částic a odstraňování starých. Poslední třída komponenty obsahuje definici samotných částic (`ParticleVertex`). Veškeré operace s částicemi (nejen vykreslování, ale i jejich pohyb, případnou změnu velikosti a tak dále) se dále odehrávají v příslušném shaderu. Díky tomu je zátěž na CPU minimální.

6.4.2 Integrace do projektu

Pro správu všech částicových systémů v herní úrovni jsem vytvořila třídu `ParticleSystemsManager`. Ta má na starosti jejich vytváření, aktualizaci i odstraňování. Všechny částicové systémy se odstraňují až po konci herní úrovně, nicméně vykreslují se kvůli zvýšení efektivity pouze částicové systémy, které jsou ve viditelné vzdálenosti od herního hrdiny. Každý částicový systém také musí mít svůj vlastní časovač pro vypouštění nových částic. Tato činnost se totiž neděje automaticky.

Ve hře NanoHeal jsou v současné době vytvořené dva typy částicových systémů. Jedním je `TrailParticle`. Ten vytváří efekt trysky za nanorobotem díky nastavení postupného zmenšování částic, nulové rychlosti částic a nulové náhodnosti při vypouštění nových částic. Vzhledem k různému barevnému ladění efektu bylo nutné použít bílou texturu a zabarvení nastavovat programově při vytváření efektu.

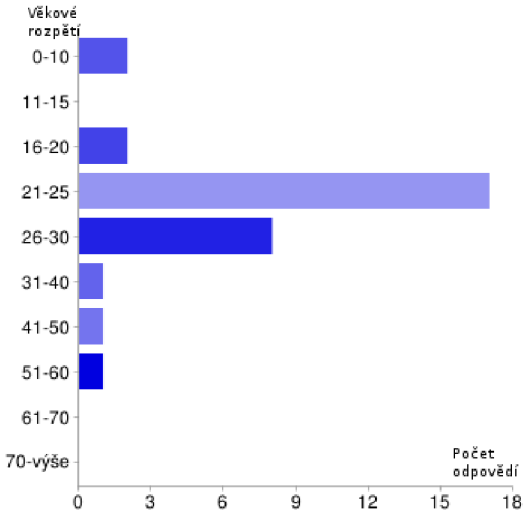
Dalším efektem použitým ve hře jsou bublinky. Ty mají dokreslovat atmosféru světa v krevním řečišti. U nich již bylo možné si lépe vyhrát s různými nastaveními. Mohou tedy rotovat, mají různou dobu života a také náhodnou rychlost.

Shader pro vykreslování částicových efektů bylo nutné doplnit o možnost renderování hloubky pro správné chování v Depth of Field efektu.

7 Výsledky a reakce uživatelů

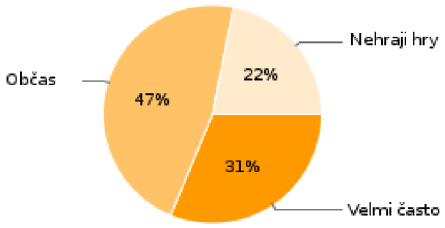
Pro účely vyhodnocení výsledků projektu jsem s pomocí Google dokumentů vytvořila internetový dotazník. Jeho přesné znění je možné nalézt v příloze 2 k tomuto dokumentu. Spolu s odkazem na něj jsem uživatelům rozesílala soubor obsahující verzi hry NanoHeal s pouze první tréningovou úrovní odemčenou. Všechny hráče jsem požádala, aby hru dohráli alespoň k první úrovni nemoci malárie (tedy aby dokončili alespoň tři úrovně ze dvanácti).

Dotazník s celkovým počtem deseti otázek zodpovědělo celkem 32 uživatelů různých věkových kategorií (viz Ilustrace 7.1). Snažila jsem se jej koncipovat tak, aby nebyl příliš časově náročný na vyplnění a aby zhodnotil všechny aspekty klíčové pro úspěch hry.



Ilustrace 7.1: Graf zobrazující počet respondentů v jednotlivých věkových kategoriích

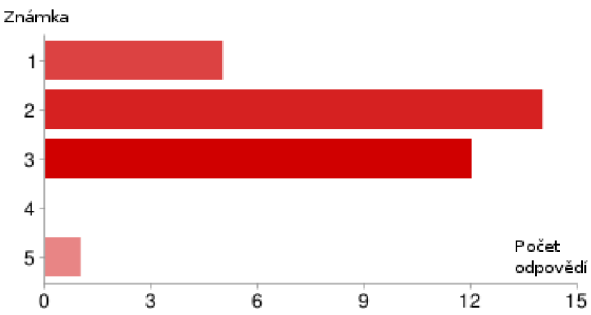
Kromě věku je ke správnému vyhodnocení důležitý dotaz na zkušenosti s hrami. Na ilustraci 7.2 vidíme, že se dotazníkového šetření zúčastnili jak hráči, kteří hrají několikrát týdne (odpověď „Velmi často“), tak i uživatelé, kteří hry vůbec nehrají. Nejčastější však byli příležitostní hráči, kteří hrají několikrát měsíčně.



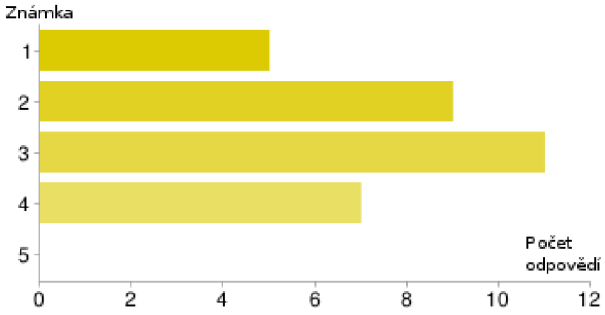
Ilustrace 7.2: Graf zobrazující četnost hraní her u respondentů

7.1 Klíčové aspekty hry

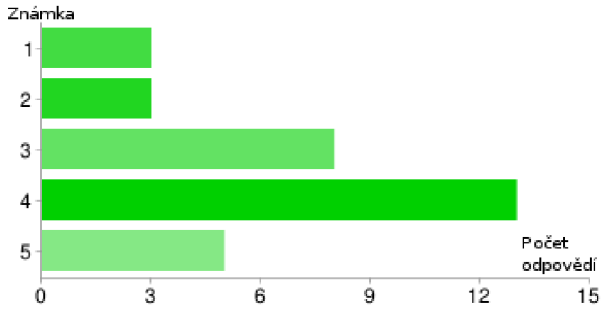
Další tři dotazy se týkaly ohodnocení klíčových aspektů hry známkami „jako ve škole“ (nejlepší tedy byla známka 1 a nejhorší 5). Klíčovými aspekty jsou vizuální vzhled, zábavnost hry a také její obtížnost (viz Ilustrace 7,3, 7.4, 7.5).



Ilustrace 7.4: Graf ohodnocení grafického vzhledu hry



Ilustrace 7.3: Graf ohodnocení zábavnosti hry



Ilustrace 7.5: Graf ohodnocení obtížnosti hry

Z ilustrace 7.4 je patrné, že téměř všichni hráči považují hru za graficky dobrou a dokonce 60% uživatelů oznámkovalo tento aspekt dvěma nejlepšími známkami. Zde se nejspíše vyplatil risk s netradičním, originálním grafickým vzhledem hry a celková grafická jednotnost titulu. Nedá se také vypozorovat žádný rozdíl mezi zkušenými hráči a ne-hráči v hodnocení vzhledu hry. V poznámkách k vyplněnému dotazníku se ale objevilo i několik negativních reakcí na ostrost obrazu a přílišné použití červené barvy. Tyto reakce však mohou souviset s nastavením a kvalitou zobrazovacího zařízení.

V případě hodnocení hratelnosti již výsledky nejsou tak jednoznačné. Méně než 50% hráčů označilo tento aspekt dvěma nejlepšími známkami. Věřím, že tyto výsledky značně souvisí s hodnocením obtížnosti hry. Více než polovina respondentů totiž hru považuje za velmi obtížnou (dvě nejhorší známky).

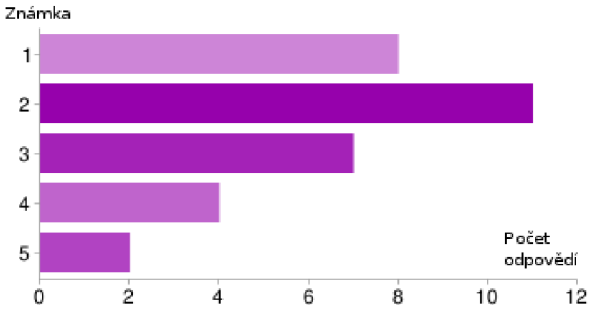
Překvapilo mě, že známky označující hru za velmi obtížnou se objevují jak u častých hráčů, tak u ne-hráčů. Ještě větším překvapením pak bylo, že u častých hráčů se nevyskytlo ani jedna známka lepší než „2“, ale u občasných hráčů a ne-hráčů se tyto známky objevily. Tyto výsledky ukazují k rostoucímu trendu jednodušších komerčních her.

V dnešní době totiž vzhledem k masovému rozšíření her je třeba zaujmout i méně náročnější a méně zdatné hráče, které by neúspěch či neporozumění herním mechanikám od hraní (a potažmo koupě) hry odradilo. Proto je třeba uživatele od začátku hry takzvaně „vodit za ručičku“ a podrobně a názorně mu vše předvádět a opakovat. Již se v rámci komerčních her s vysokým rozpočtem téměř nesetkáme s hrami, ve kterých je jednu úroveň potřeba hrát mnohokrát, než se ji hráči podaří překonat. Častí hráči jsou pak nejspíše na tento trend zvyklí, a proto jim NanoHeal připadal velmi obtížný.

Zábavnosti hry NanoHeal nejspíše škodí i další fakt, který se často objevil v poznámkách k vyplněnému dotazníku. Jedná se o nepřehlednost v orientaci mezi jednotlivými hloubkami herní scény. Tento problém byl během vývoje několikrát adresován (viz podkapitola 5.2.5), nicméně stále je zde prostor pro vylepšení. Nepřehlednost může být také způsobena samotným neobvyklým konceptem hry. Je možné, že kdyby hráči věnovali hře více času, tak by si na netradiční pohyb v prostoru zvykli, stejně jako by se naučili vzhled jednoduchých objektů a neměli by problém je rozeznat.

7.2 Naučnost

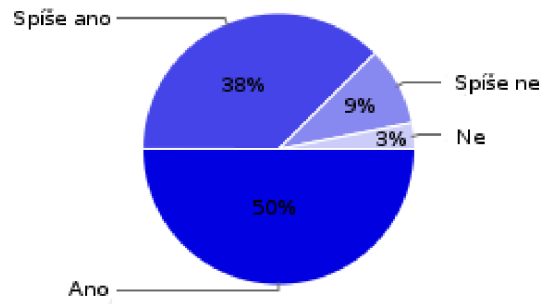
Vzhledem k tomu, že se jedná o výukovou hru, bylo nutné zjistit, jak na uživatele působí NanoHeal z tohoto pohledu. Nejprve jsem zjišťovala, zda-li hráče zaujal hlavní příběh hry. Ten sleduje doktora, vynálezce technologie, s jejíž pomocí hráč léčí pacienty.



Ilustrace 7.6: Graf zobrazující ohodnocení zajímavost příběhu hry

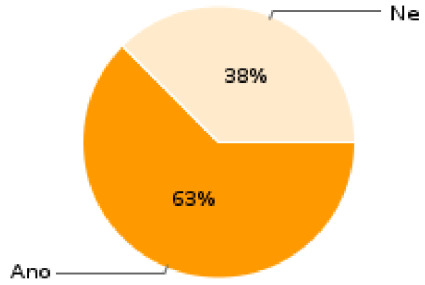
Z ilustrace 7.6 vidíme, že reakce na zajímavost příběhu jsou převážně kladné, i přesto, že téma hry nemusí být zajímavé pro každého. Téměř 60% respondentů ohodnotilo zajímavost příběhu dvěma nejlepšími známkami. Myslím, že tento výsledek mohu považovat za úspěch.

Za velký úspěch považuji výsledek další otázky dotazníku. Ta se týká názoru hráčů na vliv hry ke zvyšování povědomí o zobrazených vážných nemocech (viz Ilustrace 7.7). Drtivá většina uživatelů odpověděla „Ano“ a „Spíše ano“ (celkem 88%). Hlavní cíl, jímž bylo upozornění na tyto vážné nemoci, byl tedy podle odpovědí respondentů splněn.



Ilustrace 7.7: Graf zobrazující názor uživatelů na vliv hry na zvyšování povědomí o vážných nemocích

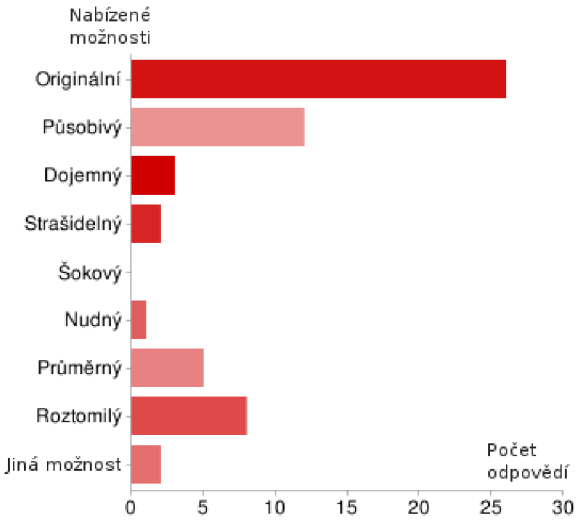
Trochu horší jsou výsledky pro dotaz, zda uživatelé hledali další znalosti ve vestavěné databázi (viz Ilustrace 7.8). Téměř 40% se totiž do herní databáze vůbec nepodívalo, což považuji za relativně velký počet uživatelů. Důvodem může být fakt, že hráč při hře není vůbec upozorňován na odblokovávání vstupů v databázi. Její existence si tak vůbec nemusí všimnout.



Ilustrace 7.8: Graf zobrazující kolik procent uživatelů se podívalo do herní databáze

7.3 Celkový dojem ze hry

V závěru dotazníku jsem požádala respondenty o zvolení celkového dojmu ze hry. Měli k dispozici několik nabízených možností (každý uživatel mohl zvolit libovolný počet z nich) a dále textové pole pro jakékoliv další poznámky. Na ilustraci 7.9 je možné vidět, že velký počet (81%) všech respondentů zvolilo možnost „Originální“. Této volby si od respondentů velmi cením, protože v dnešní době, kdy „téměř cokoliv“ již existuje, je velmi těžké vytvořit něco, co lidé považují za originální.



Ilustrace 7.9: Graf zobrazující jaký dojem v uživatelích hra vyvolala

Velmi častou kombinací spojení bylo „Originální“ a „Působivé“ či „Originální“, „Působivé“ a „Roztomilé“. Především častý výskyt možnosti „Roztomilý“ mě překvapil. Někteří testeři během vývoje totiž hru označovali za děsivou. Uživatelé, kteří nenašli vhodnou možnost v nabídce volili například označení „Naučný“ či „Nepřehledný“.

V závěrečném prostoru pro poznámky se kromě již zmíněné nepřehlednosti a kritiky barevného ladění objevily i návrhy na vylepšení ovládání. Některým hráčům se zdály reakce na změny pohybu příliš pomalé a dalším vadila nemožnost držet šipku jedním směrem pro pokračující pohyb tímto směrem.

Z výsledků dotazníkového šetření vyplývá několik nedostatků hry NanoHeal, které by bylo možné odhalit dříve, kdyby bylo provedeno důkladné testování hry – tzv. „Focus testing“. Ten je u současných titulů nezbytnou součástí vývojového cyklu. Jedná se o delší nepřetržité hraní daného titulu několika testery. Při něm je pozorují vybraní členové vývojového týmu a pečlivě sledují reakce hráčů a způsob jejich hraní. O důležitosti správného testování pojednává například [24].

V případě hry NanoHeal bylo testování z časových důvodů bohužel značně zanedbáváno. Nicméně i přesto považuji projekt za velice úspěšný a věřím, že po dopracování několika zmíněných nedostatků by zaujal velké množství potenciálních hráčů.

8 Závěr

Během práce na diplomovém projektu jsem se neustále učila nové věci a řešila různorodé problémy. Vývoj her je totiž širokou problematikou, kde se programátor musí zabývat nejen programováním grafiky, ale i zvuku, herní logiky, fyziky a mnoha dalšími odvětvími. Nestačí také jen programovat, pro hru je třeba vytvořit grafický herní obsah. Projekt mi tak přinesl mnoho v budoucnu jistě užitečných zkušeností.

Při tvorbě diplomového projektu jsem se seznámila s problematikou vývoje pro XNA a některými existujícími výukovými hrami. Navržená hra NanoHeal se zabývá závažnými světovými problémy ve zdravotnictví. Hráč se dostává do role nanorobota a má za úkol co nejšetrněji vyléčit pacienta. Po dokončení vývoje hry proběhlo dotazníkové šetření, jehož výsledky jsou popsány v kapitole 7.

Výsledným produktem této práce je dokončená aplikace hratelná na počítačích s operačním systémem Windows i na herních konzolách Xbox 360. Hra se zabývá třemi nebezpečnými nemocemi (malárií, infarktem a AIDS) a byla vyhlášena nejlepší hrou kategorie herní design v národním kole soutěže Imagine Cup. V mezinárodním kole se titulu NanoHeal podařilo postoupit z téměř sedmi set účastníků do sto padesáti nejlepších. Výsledky dalšího mezinárodního kola nejsou bohužel v době odevzdání této práce ještě známy.

NanoHeal je samozřejmě možné dále vylepšovat. Především kvůli nedostatku systematického testování hra trpí nevyváženou obtížností. Mnoha testujícím uživatelům se hru bohužel z tohoto důvodu nepodařilo dohrát do pozdějších úrovní. Z dotazníkového šetření také vyplynulo, že hře by prospělo intuitivnější ovládání a větší uživatelská přívětivost při vysvětlování herních mechanik přímo ve hře.

Vzhled hry by obohatily další post-processingové efekty či vylepšení implementace rozmazání při efektu hloubky obrazu (například na gaussův blur). Mnoha testujícím uživatelům se také zdála hra nepřehledná kvůli podobnosti interaktivních objektů ve hře. Tento problém jsem během vývoje zkoušela vyřešit různými způsoby (viz kapitola 5.2.5), nicméně stále je zde prostor k vylepšení.

Ráda bych hru po dopracování některých výše zmíněných nedostatků poskytla co nejvíce zájemcům skrze webovou stránku. Věřím také, že bude užitečným příspěvkem do mého projektového portfólia. Vývoji her se totiž chci věnovat i nadále.

S výsledným projektem jsem spokojena a věřím, že splnil veškeré body zadání i všechny klíčové stanovené požadavky. Uživatelé hru často označovali jako originální a zajímavou, což považuji za velký úspěch. Myslím, že přinese zábavu i nové znalosti či aspoň touhu po vědění v oblasti nejen zdravotnictví mnoha hráčům. Hra by také mohla najít své využití například ve školství.

Literatura

[1] WWW stránky: XNA Creators Club [online]. http://creators.xna.com/en-US/ [cit. 2010-01-03].

[2] WWW stránky: Microsoft Development Center [online]. http://msdn.microsoft.com/en-us/library/bb195055.aspx [cit. 2010-05-13].

[3] WWW stránky: Blender Artists [online]. http://blenderartists.org/forum/showthread.php?t=119783 [cit. 2010-01-03].

[4] Storm R., Modeling for XNA with Blender [online]. http://www.stromcode.com/2008/03/10/modelling-for-xna-with-blender-part-i/ [cit.2010-01-03].

[5] WWW stránky: Imagine Cup [online]. http://imaginecup.com/default.aspx [cit. 2010-01-03].

[6] Kapp K.: Kapp Notes [online]. http://karlkapp.blogspot.com [cit. 2010-01-03].

[7] WWW stránky: Nobel prize [online]. http://nobelprize.org/educational_games/ [cit. 2010-01-03].

[8] WWW stránky: PowerUp The Game [online]. http://www.poweruptheGame.org/home.html [cit. 2010-01-03].

[9] WWW stránky: FreeGame [online]. http://www.freegame.cz/ [cit. 2010-01-03].

[10] WWW stránky: Unreal Technology [online]. http://www.unrealtechnology.com/ [cit. 2010-01-03].

[11] WWW stránky: Wikipedia: The Free Encyclopedia [online]. http://en.wikipedia.org/wiki/Microsoft_XNA [cit. 2010-01-03].

[12] WWW stránky: MSXbox World [online]. http://www.msxbox-world.com/xbox360-specification.php [cit. 2010-01-03].

[13] Hargreaves S., Game Programming with XNA Framework [online]. http://www.talula.demon.co.uk/blogindex.html [cit. 2010-01-03].

[14] WWW stránky: BlenderWiki [online]. http://wiki.blender.org/index.php/Extensions:Py/Scripts/Manual/Export/FBX [cit. 2010-01-03].

[15] Strothotte T., Schlechtweg S.: Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation. Morgan Kaufmann, San Francisco. ISBN 1-55860-787-0.

[16] Reynolds C.: Stylized Depiction in Computer Graphics [online]. http://www.red3d.com/cwr/npr/ [cit. 2010-01-03].

[17] WWW stránky: International Symposium on Non-Photorealistic Animation & Rendering [online].
http://www.cs.rug.nl/svcg/npar2009/ [cit. 2010-01-03].

[18] WWW stránky: Jeux Video [online].
http://www.jeuxvideo.com [cit. 2010-01-03].

[19] WWW stránky: Wikipedia: The Free Encyclopedia [online].
http://en.wikipedia.org/wiki/History_of_video_game_consoles_(seventh_generation)
[cit. 2010-05-12].

[20] Wilhelmsen P.: SXNA Shader Programming: Tutorial 20 – Depth of Field [online].
http://digierr.spaces.live.com/blog/cns!2B7007E9EC2AE37B!662.entry [cit. 2010-05-12].

[21] Miles R.: Microsoft XNA Game Studio 3.0: Learn Programming Now!.
ISBN 978-0-7356-2658-4.

[22] Grootjans R.: XNA 2.0 Game Programming Recipes, A Problem-Solution Approach.
ISBN 978-1-59059-925-9.

[23] WWW stránky: WiX XNA Installer [online].
http://xnainstaller.codeplex.com/ [cit. 2010-05-13].

[24] WWW stránky: Gamasutra [online].
http://www.gamasutra.com/view/news/27390/InDepth_How_Planning_Derailed_Playtesting_Redeemed_Uncharted_2.php [cit. 2010-05-19].

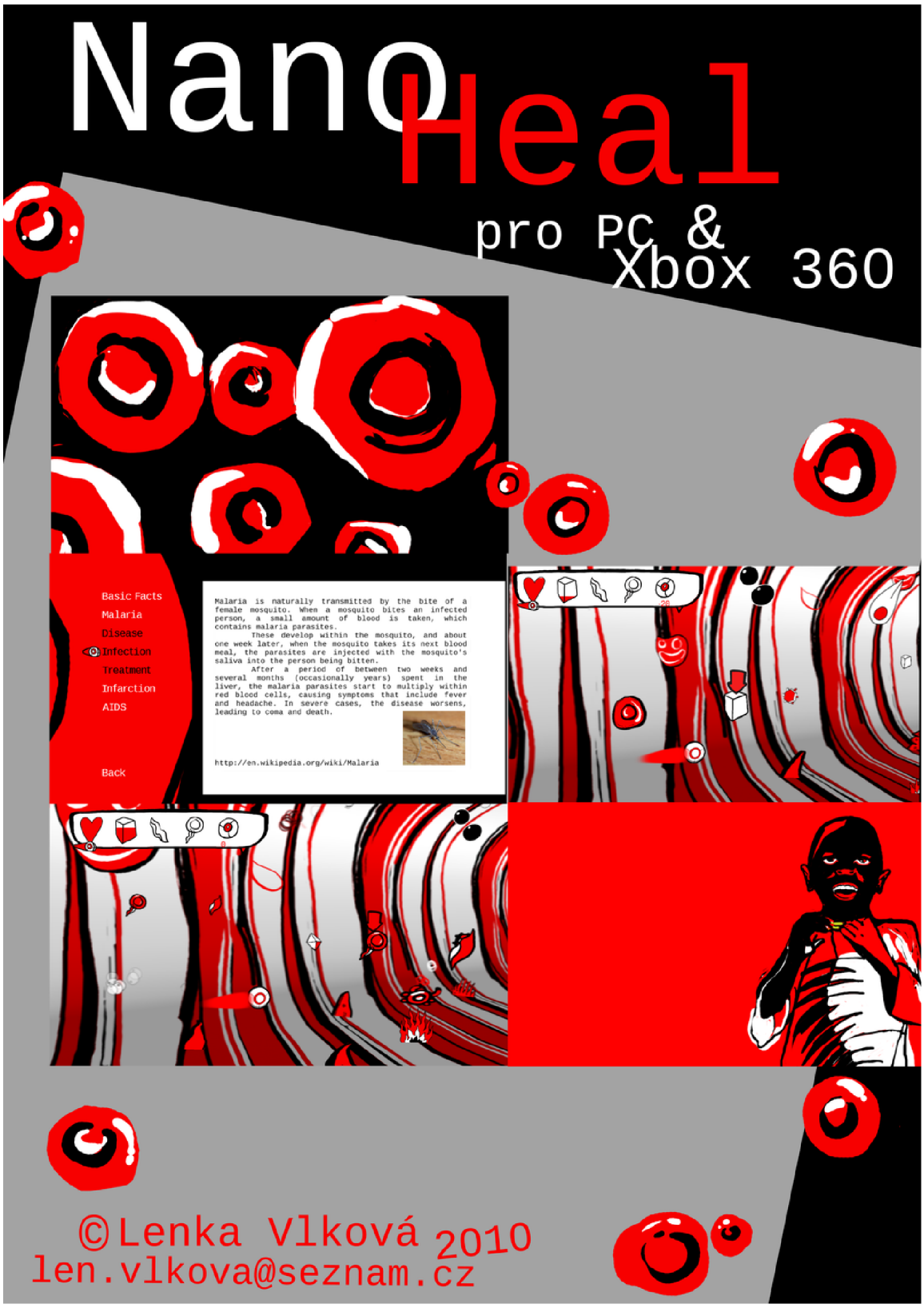
Seznam příloh

Příloha 1. Obsah přiloženého CD

K této práci je přiložené CD, které obsahuje následující adresářovou strukturu:

/ Docs	
design_spec1.4.pdf	Design dokument
xvlkov00_DP.odt	Zdrojový kód k tomuto dokumentu
xvlkov00_DP.pdf	Tento dokument ve formátu .pdf
/NanoHeal-game	
readme.txt	Pokyny k instalaci
manual.pdf	Manuál ke hře NanoHeal
/Full_XNA	
NanoHeal_Xbox360.ccgame	Verze pro Xbox 360
NanoHeal_PC.ccgame	Verze pro PC
XNAGS31_setup.exe	XNA Game Studio instalátor
/Redist_ver	
NanoHeal.msi	Instalátor PC verze pro distribuci
Settings.set	Soubor pro odblokování všech úrovní
/NanoHeal-src	
readme.txt	Návod pro kompilaci
/Dokumentace	
index.html	Programová dokumentace
/NanoHeal	
NanoHeal.sln	Projekt pro Visual Studio
/Pics	
Plakat.tif	Plakát A2
NanoHeal0.jpg – NanoHeal4.jpg	Obrázky ze hry
Trailer.wmv	Upoutávka na hru

Příloha 2. Plakát reprezentující práci



Příloha 3. Dotazník

Otázky označené znakem * jsou povinné.

1. Zvolte prosím váš věk *
Možnosti: 0-10, 11-15, 16-20, 21-25, 26-30, 31-40, 41-50, 51-60, 61-70, 70-výše

2. Jak často hrajete hry? *
Možnosti: Velmi často (několikrát týdně), Občas (několikrát měsíčně), Nehraji hry

3. Ohodnoťte grafiku hry od 1 do 5 * (1 líbí se mi nejvíce, 5 nejméně)
Možnosti: 1, 2, 3, 4, 5

4. Ohodnoťte zábavnost hry * (1 - velmi zábavná, 5 - nezábavná)
Možnosti: 1, 2, 3, 4, 5

5. Ohodnoťte obtížnost hry * (1 lehká, 5 - těžká)
Možnosti: 1, 2, 3, 4, 5

6. Zaujal Vás příběh hry? * (1 velmi, 5 - vůbec)
Možnosti: 1, 2, 3, 4, 5

7. Myslíte si, že tato hra přispívá k povědomí o závažných nemocech? *
Možnosti: Ano, Spíše ano, Spíše ne, Ne

8. Podívali jste se do herní databáze? *
Možnosti: Ano, Ne

9. Jaký dojem ve Vás hra vyvolala? * (může být více odpovědí)

- Originální
- Působivý
- Dojemný
- Strašidelný
- Šokový
- Nudný
- Průměrný
- Roztomilý
- Other:

10. Prostor pro jakékoliv další poznámky...

Příloha 4. Příspěvek do sborníku českého národního kola soutěže Imagine Cup

Následující dokument byl vytvořen pro účast v českém národním kole soutěže Imagine Cup. V něm se náš tým Puellae s projektem NanoHeal stal vítězem. Tým má tři členy. Mým hlavním úkolem byla kompletní implementace hry a design herních úrovní. Hana Vrzáková vytvořila aplikaci pro design úrovní a Adéla Wiederlechnerová měla na starosti 2D herní obsah a grafický design. Na tvorbě dokumentu se podílel celý tým. Má za cíl shrnutí našeho projektu.

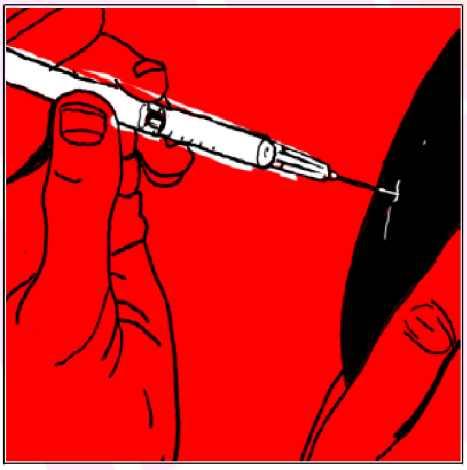
NANOHEAL

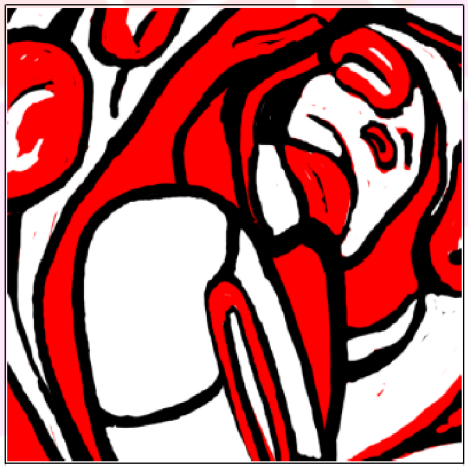
Tým: Puellae
Škola: VUT Brno Fakulta informačních technologií, UTB Zlín
Mentor: -
Řešitelé: Lenka Vlková, Hana Vrzáková, Adéla Wiederlechnerová

NanoHeal

Úvod

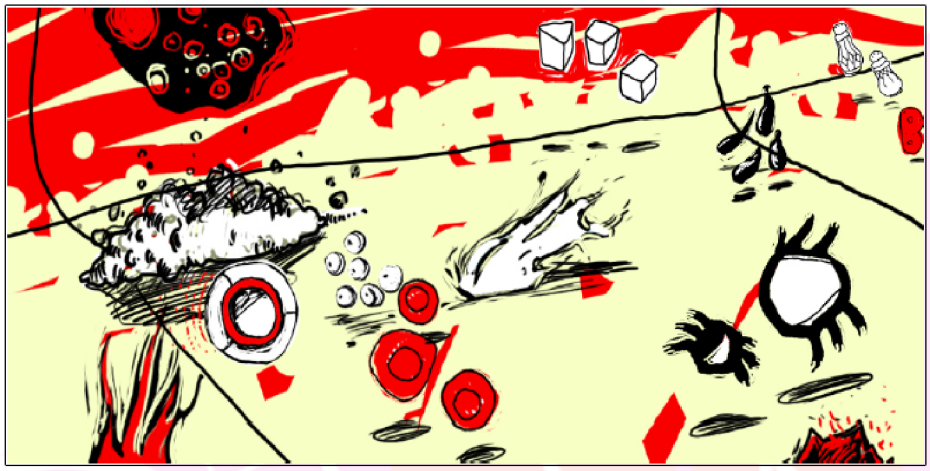
Projekt se zaměřuje na šestý z osmi celosvětových cílů OSN pro rok 2015, což je boj s malárií, HIV/AIDS a dalšími nemocemi. Snahou projektu je zvýšit povědomí o těchto závažných onemocněních formou naučné hry, v rámci které může hráč vyléčit pacienta pomocí nanotechnologie. Hra je ztvárněna stylizovanou 3D grafikou, která se nesnaží o věrné zachycení reality, nýbrž o komiksové přiblížení lidského těla a jeho součástí.





1. Myšlenka projektu

Hráč ovládá nanobota, který proplouvá lidskými žílami a tepnami. Nanobot nese lék na konkrétní nemoc a jeho úkolem je donést jej včas do centra mikrobů a virů, čímž pacienta může vyléčit. Pro svůj pohyb tělem potřebuje nanobot energii, která mu umožňuje vyhýbat se lidským tkáním a buňkám tak, aby nepoškodil pacientovo tělo. Tuto energii získává sbíráním cukrů dostupných v lidském těle. Avšak musí být velmi opatrný, aby v těle nepoškodil či neodebíral žádné další potřebné látky a tím dále neohrožoval pacientovo zdraví.



Příběh jménem NanoHeal

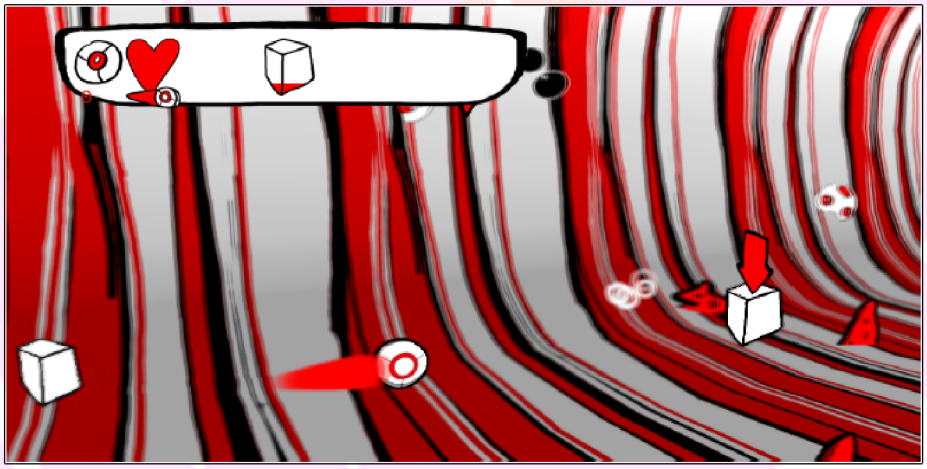
Píše se rok 2015 a nejmenovaný doktor nejmenované instituce právě odlétá směrem k divoké Africe. Jeho cílem je malý nemocný chlapec, který se nijak neliší od dalších několika miliónů chlapců - kousl ho komár a on onemocněl malárií. Lékařská péče je v této oblasti zoufale nedostupná a listina čekatelů nekonečně dlouhá. Chlapec však souhlasil s experimentální léčbou a nechává si podat nový lék. Teď už zbývá jenom čekat a doufat, že robotek NanoHeal svůj souboj proti onemocnění vyhraje.

2. Struktura projektu

Hra NanoHeal vás přenese do krevního řečiště různých pacientů všude na světě. Hra vychází z klasických arkádových titulů a je okořeněná prvky závodních her. Myslete rychle, jednejte rychleji, čas běží!

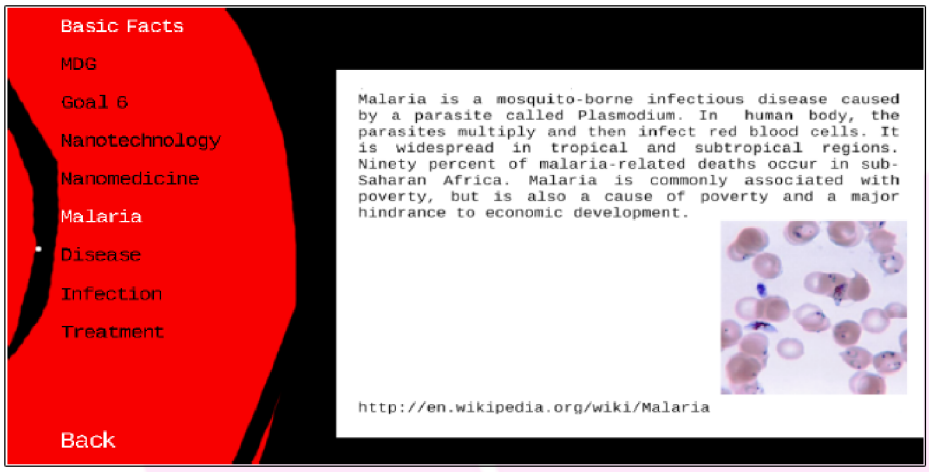
2.1 Hra

Cílem hry je vyléčit pacienta tak, že budete bojovat s nemocí, kterou trpí. Nejdříve ze všeho musíte nasbírat stopy, které vás zavedou k epicentru nákazy, podobně jako se tomu děje v imunitním systému každého člověka. To ovšem není tak jednoduché. Ani nanobot není Perpetuum Mobile - ke svému pohybu potřebuje energii. Cukry představují elementární energii volně dostupnou v lidském těle, ale i v tomto směru musíte být opatrní. Při jejím sběru je třeba dávat pozor na ostatní látky plovoucí v krevním řečišti – oslabený pacient jich nemá nazbyt a mohl by zemřít vyčerpáním dřív, než jej vyléčíte. Když se vám povede nasbírat dost indicií a dostat se až k jádru nemoci, objeví se před vámi ten opravdový protivník a boj může začít. Váš nanobot nese protilátku, která soupeře zaručeně zničí, musíte jej ale zasáhnout a zároveň dávat pozor, abyste měli dostatek energii k letu. Není to jednoduché, zvláště když neustále dáváte pozor, abyste omylem nepoškodili pacientovo tělo tím, že zničíte nějakou ze zdravých buněk.



2.2 Databáze

Porazit onemocnění se zdá snadné, jak je tomu ale doopravdy? Co je to nanotechnologie a jak fungují nanoboti? Kolik lidí ročně onemocní malárií a co to vlastně je? Všechny tyto otázky a zvláště pak odpovědi najdete v herní databázi, která vám odemkne malou část svých znalostí vždy po zápasu s nepřítelem.



2.3 Použité technologie

.NET Framework 3.5

XNA Game Studio 3.1

Adobe PhotoShop

Blender

Závěr

Projekt NanoHeal je unikátní hra, která vás vtáhne do mikrosvěta krvinek a nanobotů. Skrze příběhy a dech beroucí grafiku poznáte původce největších příkoří, které sužují zemi, a dostanete jedinečnou příležitost jim v tom zabránit. Představme si spolu svět, kdy nanotechnologie řeší věci neřešitelné. Představme si svět NanoHeal!