VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMU

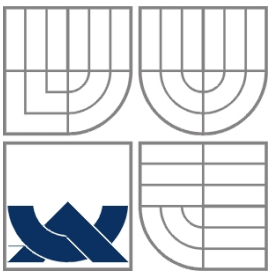
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RETROSPEKTIVNÍ ARKÁDOVÁ HRA

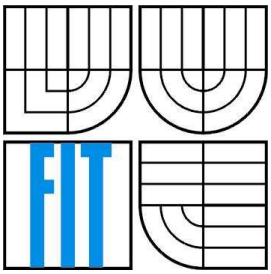
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN TRČKA
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RETROSPEKTIVNÍ ARKÁDOVÁ HRA

RETROSPECTIVE ARCADE GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN TRČKA

VEDOUCÍ PRÁCE
SUPERVISOR

ING.PETR POSPÍCHAL

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá vývojem počítačové hry inspirované historickým Arkanoidem. První část práce se zaobírá historii počítačových her a počítačové grafiky. Dále navazuje popis grafických knihoven, jakými jsou DirectX a OpenGL. V druhé části práce je rozebrán návrh aplikace a využívaný kolizní systém. Poté práce popisuje implementaci v jazyce C++ s použitím objektového návrhu a knihoven OpenGL a SDL. Závěr je věnován principu testování hry.

Abstract

This bachelor's thesis describes the evolution of computer, which is inspired of historical game called Arkanoid. The first part of the thesis describes history of computer games and conputer graphics. Next section details the graphics libraries as DirectX and OpenGL.The second part of the thesis describes the plan of the game and collisional system. Then the thesis includes description of implementation, where are used object oriented design, library OpenGL and library SDL in C++. The end of thesis is devoted to testing of game.

Klíčová slova

OpenGL, SDL, C++, Počítačová hra, Počítačová grafika, Kolize, Arkanoid

Keywords

OpenGL, SDL, C++, Computer game, Computer graphics, Collision, Arkanoid

Citace

Martin Trčka: Retrospektivní arkádová hra, bakalářská práce, Brno, FIT VUT v Brně, 2010

Retrospektivní arkádová hra

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petra Pospíchala. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Trčka
17.5.2010

Poděkování

Rád bych na tomto místě srdečně poděkoval Ing. Petru Pospíchalovi, který si vždy našel čas na konzultaci a poskytoval cenné a odborné rady při vypracování této práce.

Obsah

1 Úvod

V současné uspěchané době, je více než nutné najít si čas na odpočinek, uvolnění, popřípadě zábavu. Minulost ukázala, že lidem k zábavě a odpočinku stačil obyčejný kopací míč, poutavá knížka nebo posezení v příjemném podniku. Rapidní vývoj elektroniky v poslední době umožnil přesun zábavy i do virtuálního, počítačového světa.

Moderní herní průmysl už dávno nevytváří hry s jednoduchými principy a hříčkami, ale díky současným audiovizuálním možnostem a moderním grafickým efektům, je možné vytvořit plně interaktivní hry, které mohou vzdělávat, ale i vylepšovat motoriku a rozhodování. Vývoj dnešních moderních her již není otázkou několika dnů, nebo týdnů, ale zpravidla se jedná o měsíce, ne-li roky.

Současné hry jsou také atraktivní z pohledu programovaní. Je to způsobeno tím, že jejich realizace se dotýká řady oborů, od počítačové grafiky přes umělou inteligenci a algoritmy až po simulaci fyziky. To často představuje programátorskou výzvu.

Bakalářská práce se zabývá vývojem a tvorbou počítačové hry inspirované historickým Arkanoidem. Začátek práce je věnován historii a rozdělení počítačových her do kategorií s popisem základních principů. Závěr této kapitoly bude patřit současné situaci počítačových her a jejich postavení ve společnosti. Následující kapitoly mají za úkol seznámit čtenáře se základy počítačové grafiky a grafických knihoven, jakými jsou OpenGL a DirectX. Další část práce obsahuje postup při navrhování samotné hry a způsob její implementace.

Závěrem bakalářské práce je zhodnocení celé hry, sbírání názorů od uživatelů a diskuse o možném rozšíření a dalším vývoji.

1.1 Členění práce

Celá práce je rozdělena do kapitol s názvy podle tematických okruhů.

Druhá kapitola bakalářské práce seznamuje čtenáře s pojmem počítačová hra. Je zde diskutována jak samotná historie počítačových her, tak i jejích postupný vývoj. Dále se kapitola věnuje rozdělení počítačových her do kategorií neboli žánrů.

Následující kapitola se zabývá počítačovou grafikou. Popisuje se zde historie počítačové grafiky a její základní dělení na rastrovou a vektorovou grafiku.

Kapitola 4 seznamuje čtenáře s dostupnými grafickými knihovnami, jakými jsou DirectX a OpenGL. V každé podkapitole je jednotlivá knihovna rozebrána s pohledu historie a vývoje. Další odstavce podkapitol diskutují také o výhodách a nevýhodách popsaných knihoven.

Kapitola 5 prezentuje čtenáři návrh celé aplikace. Popisuje zvolené programové řešení a postup při vytváření návrhu pro tvorbu počítačové hry. Dále kapitola obsahuje návrh herního systému a princip počítání kolizí pomocí sférického rozdělení objektů.

Implementační část, kapitola 6, plně navazuje na návrh. Jsou zde rozebrány jednotlivé třídy a diskutovány jejich hlavní prvky.

Kapitola 7 popisuje principy testování s využitím hodnocení od uživatelů. Testování probíhalo pomocí dvou základních metod, black-box a white-box[14].

Poslední kapitola 8 shrnuje dosažené výsledky a přibližuje směr dalšího vývoje aplikace.

2 Počítačová hra

Jak už popisuje úvod, počítačové hry jsou v dnešní době obrovským fenoménem zasahující do mnoha oblastí. Ať už pozitivně nebo negativně. Každá počítačová hra má svůj vlastní svět, ovládán pomocí klávesnice, myši a různých herních ovladačů. Začíná se také používat ovládání přes hlasové příkazy za asistence mikrofonu. Díky ovládacím periferiím je hráč přímo v kontaktu s okolním virtuálním světem a může ho ovlivňovat, jak uzná za vhodné. Musí plnit určité úkoly, které jsou omezené časovým limitem, nebo mají základ v logické hříčce.

2.1 Historie počítačových her

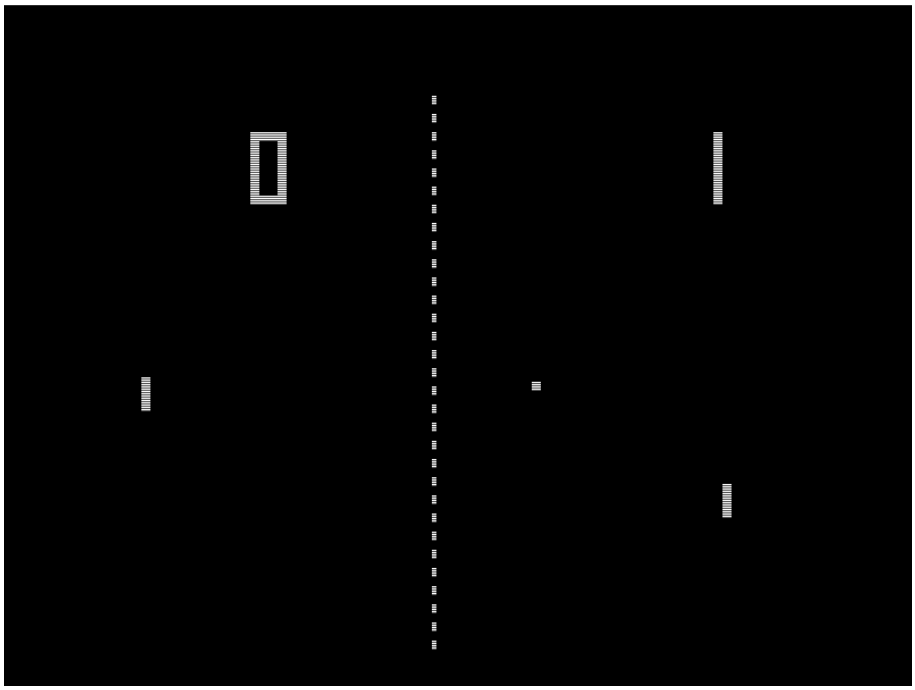
Počátky počítačových her se začaly psát ve 40. a 50. letech 20. století, přesněji v roce 1947, kdy fyzikové Thomas Goldsmith a Estl Mann vyvinuli aplikaci, využívající osciloskop a katodové trubice. Princip hry byl primitivní. Pomocí trajektorie světelných stop měly být zasáhnuty cíle zobrazeny na zobrazovacím zařízení. O pár let později, v roce 1951, byl představen digitální počítač Ferranti NIMROD. Šlo o první počítač navržený speciálně pro hru s názvem NIM, což je jednoduchá logická hra pro dva hráče. Pro grafický (*vizuální*) výstup byl použit panel se žárovkami. Roku 1952 Alexandr S. Douglas naprogramoval v rámci práce na téma interakce člověka s počítačem aplikaci OXO, určenou pro počítač EDSAC. Jednalo se o jednoduchou variantu piškvorek, kde se vše zobrazovalo na CRT displeji s rozlišením 35 x 16 pixelu[3].

Roku 1958 vytvořil William Hoganbotham aplikaci s názvem Tennis for Two, určenou primárně pro zabavení návštěvníků Národní laboratoři Brookhaven v New Yorku. Celá hra běžela pomocí analogového počítače a osciloskopu. Jednalo se o jistou variaci tenisu, kde byl na osciloskopu zobrazen z profilu tenisový kurt. Hra obsahovala síť a míček s gravitačními vlastnostmi[3][4].

První čistě počítačovou hrou se stala aplikace Spacewar!. Roku 1962 ji vyvinuli studenti Massachusettského technologického institutu pro počítač DEC PDP-1. Hra byla cílena pro dva hráče ovládající každý svojí vesmírnou loď. Uprostřed herní plochy se nacházela hvězda, jež svou gravitací ovlivňovala pohyby lodí, nicméně nikoli jejich střel – na to nestačil výpočetní výkon[4].

Po dlouhém téměř desetiletém čekání přichází v 70. letech první legendy ze světa her. Vše nastartovala dvojice Nolan Bushnell a Ted Dabney založením společnost Atari v roce 1972 a stejný rok také vydali první masově rozšířenou hru Pong, která připomínala stolní tenis. Atari dokázala prodat téměř 19 000 herních zařízení s Pongem, což se dá považovat za obrovský úspěch a vzestup automatových her[4].

První vážná diskuse a kontroverze týkající se násilí v počítačových hrách vypukla již v roce 1976, kdy byla vydána hra Death Race, jejímž cílem bylo autem přejíždět protivníky. Tato hra zvýšila vědomí veřejnosti o videohrách. Ovšem oblíbenost arkádových hříček (*viz. 2.2*) stále rostla a projevila se na fenomenálním úspěchu hry Space Invaders, kteří spatřili světlo světa roku 1978 v japonské společnosti Taito. Hra podtrhla svůj úspěch i díky zápisu do Guinessovy knihy rekordů jako neúspěšnější arkádová hra. Příchod legend završil roku 1980 Pac-Man od společnosti Namco. Pac-Man se ihned stal fenoménem a novou ikonou světa počítačových her[4].



Obrázek 2.1: Hra Pong[4]

80. léta se nesla ve znamení herních systému, jako byly ZX Spectrum nebo Commodore 64. Herní průmysl začíná ekonomicky vzkvétat, vzniká mnoho nových vývojářských studií. Za zmínku stojí založení Electronic Arts v roce 1982, což je v současné době jedna z předních vydavatelských společností pro počítačové a konzolové hry. Objevují se nové herní žánry a začíná se používat pseudo-3D zobrazení herního světa nebo dokonce čistě 3D grafika. Jedna z prvních aplikací tohoto druhu se stala hra Battlezone využívající vektorovou grafiku nebo 3D Monster Maze z roku 1982. Atraktivní hrou se staly závody Pole Position využívající pseudo-3D grafiku[1]. Velký rozmach v 80. letech zaznamenaly přenosné herní konzole (*handheldy*). Rozšíření téhle kategorie herních zařízení má za následek hra Tetris. Tato hra vychází roku 1985 a ovládla tuto kategorii. Nyní existuje v nekonečně mnoha variacích a je populární až dodnes. Zlom přišel roku 1989, kdy americká firma

[1] Neboli 2.5D grafika sestavena z 2D obrazů a správně nastavené projekce

Broderbund Software vydala legendární hru Prince of Persia, jejž se stala hitem a předlohou pro mnoho dalších projektů.

Obrázek 2.2: Hra Pole Postion s pseduo-3D grafikou[4]

Během 90. let se počítačové hry postupně změnily v hlavní zábavu a rozpočty herních trhů se vyšplhaly do závratných výšin. Z části za tímto úspěchem stál i nezadržitelný technologický rozvoj, který napomohl tomu, aby hry vypadaly lépe a působily věrohodněji. Rozvíjí se téměř všechny herní žánry (*viz. 2.2*). Pracuje se již s plně 3D grafikou, která demonstruje současné technologické možnosti. Mnoho současných her má zde zapuštěné kořeny. Vznikají obří studia jako je Westwood Studios, Blizzard Entertainment, id Software. Tyto studio směřující vývoj počítačových her kupředu. V letech 1992 – 1995 vychází kultovní tituly, jako je Warcraft: Orc & Humans (*1994*), případně série Command & Conquer (*1995*) nebo Sim City 2000 (*1993*). Začíná obrovský rozvoj akčních her jako Wolfenstein 3D (*1992*), který se stal první úspěšnou 3D akcí a odstartoval celou éru jeho následovníků. Roku 1993 vychází také legenda všech akčních her. Doom od id Software s revolučním grafickým kabátem. V roce 1996 se objevila grafická karta Voodoo od 3dfx Interactive jako první dostupný 3D akcelerátor pro domácí počítače. Vše plně demonstrovala společnost id Software se svou herní sérií Quake. Ten svým grafickým a fyzikálním systémem položil základy několika dalším hrám. Jako první používala kompletní 3D modely. S rostoucím výkonem počítačů se prosazuje perfektní audiovizuální stránka ve všech herních žánrech a stává se základem virtuálního světa[4].

Obrázek 2.3: Hra Warcraft: Orc & Humans[4]

2.2 Žánry počítačových her

Počítačové hry se už od samého počátku řadí do různých kategorií nebo žánrů. Podle toho lze určit základní principy dané hry a její možnosti. Následující odstavce popisují základní rysy každé kategorie.

Adventura[2] je žánr, založen na zajímavém a spletitému příběhu, s mnoha hádankami a zápletkami. Hráč, jako hlavní hrdina, řeší rozličné úkoly, hádanky a pomalu rozplétá příběh s užitím své fantazie a logického myšlení, aby vyřešil virtuální patálie. Na závěr bývá zobrazena animační sekvence, pro vysvětlení a doladění celého vyřešeného příběhu. Příkladem může být česká hra Polda.

Akční počítačová hra má základ v eliminaci cílů (*nepřátel*) pomocí bojových technik. Jsou zde zobrazeny dynamické boje, násilné sekvence a obrovská škála případných zbraní. Žánr patří mezi nejvíce kontroverzní počítačové hry a v mnoha zemích je striktně omezen. Akční hry jsou v současné době náročné na výkon počítače jak z důvodu pokročilé umělé inteligence nepřátel, tak i grafickým zpracováním a vizuálním efektům. Hráč přednostně ovládá hlavní postavu, případně menší skupinku bojovníků. Jako příklad může být hra Doom 3.

Arkáda je kategorie počítačových her založena na jednoduchém a nápaditém konceptu. Arkádová hra obsahuje úrovně, které jsou omezené časem a zvyšující se obtížností. Tento druh her

[2] *Z anglického slova adventure - dobrodružství*

byl v minulosti populární na herních automatech (*viz. 2.1*). Hráč musí využívat hlavně své postřehy, hbitost prstů, rychlost logického myšlení a koncentraci. V arkádách je populární hra pro dva a více hráčů na jednom počítači zároveň. Typická arkádová hra je Arkanoid.

Logické hry lze považovat za speciální žánr, jenž má za hlavní úkol u hráče naplno zaměstnat logické myšlení. Hry jsou klidného rázu, jelikož hráč se má plně zaměřit pouze na řešení obtížných úkolů. Hlavní představitel této kategorie je česká hra Berušky.

Strategické hry mají základní rysy jako je ovládání větší skupiny objektů a případná manipulace s nimi po hrací ploše. Hráč má za úkol sledovat ekonomickou situaci své virtuální říše, starat se o stav svých jednotek, hlídat a kontrolovat území. Strategické hry rozvíjejí inteligenci a analytické myšlení hráčů, tudíž jsou kladeny za vzor ostatním počítačovým hrám. Příkladem může být série Age of Empires.

Simulátory jsou speciální žánr počítačových her. Hlavní úkol těchto her je vytvořit ve virtuálním světě totožné podmínky, které panují v realitě. Simulátory čerpají z fyzikálních, právních a přírodních zákonů, které se odráží ve virtuálním světě podobně jako v reálném světě. Jako příklad uvedeme letecký simulátor Microsoft Flight Simulator.

Hry na hrdiny [3]je druh her, kde hráč zaujme roli fiktivní postavy, kterou si podle pravidel vytvoří k obrazu svému a pomocí ní ve hře jedná. Jádro hry je procházet virtuální svět, plnit úkoly a hádanky, které vedou k rozuzlení hlavního příběhu. Hráč sbírá zkušenosti podle daných pravidel a postupem hry vylepšuje své vlastnosti. Oblíbenou hrou tohoto typu je Diablo 2.

V dnešní době se žánry již aktivně prolínají a hry můžeme popsat pomocí více kategorií dohromady. Například hra Mass Effect kombinuje akční scény s prvky hry na hrdiny.

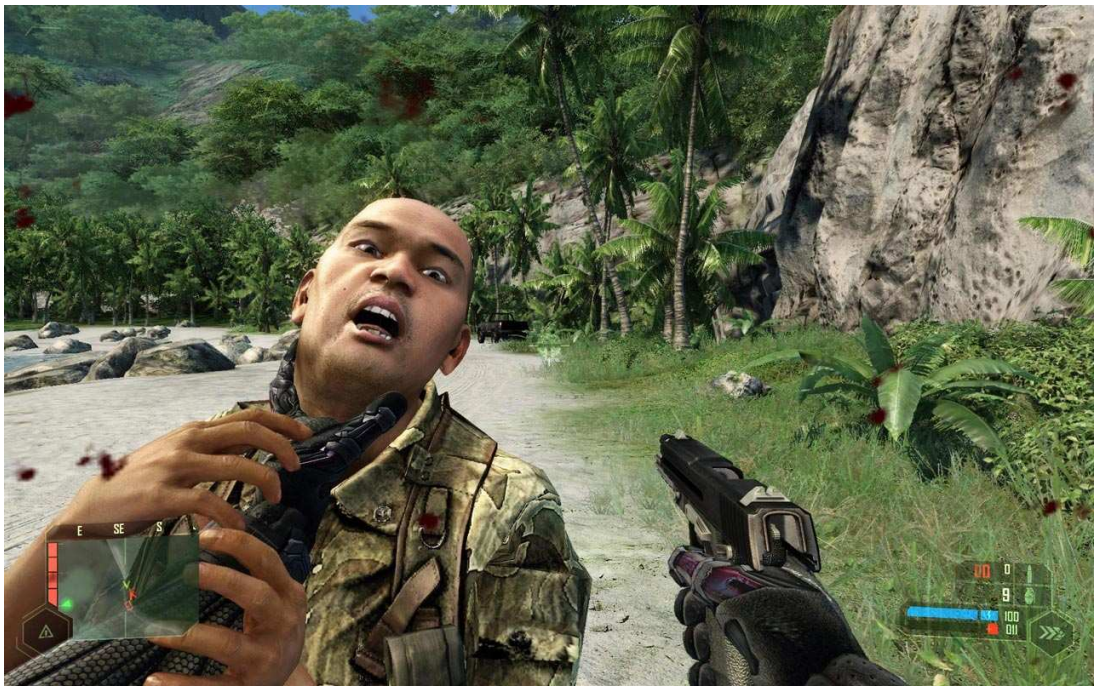
2.3 Současná situace na poli počítačových her

V současné době trh počítačových her tvoří obrovský podíl v zábavním průmyslu a stále roste. Stejně jako filmy se i počítačové hry staly napevno součásti našeho života. Vývojáři se předhání, kdo dokáže nejlépe implementovat grafické efekty, fyzikální modely a jiné doplňky, které obohacují finální aplikaci. Tento bohatý trh a vývoj technologií vytváří obrovský tlak na výrobce počítačových komponentů, kteří musí vyvíjet sofistikovanější a výkonnější grafické procesory (*dále jen GPU*[4]).

[3] Anglicky RPG – Role Playing Game.
[4] Anglicky Graphics Processing Units. Jedná se o procesor na grafické kartě, zajišťující vykreslování dat na zobrazovací zařízení (*monitor, projektor*).

Díky tomu GPU disponují silným výpočetním výkonem, který se ve hrách (*ale i v jiných oblastech*) začíná využívat v GPGPU[5] výpočtech.

V poslední době se kromě vizuální stránky začali vývojáři zaobírat i fyzikálními zákony, jako jsou efekty výbuchů, realistické deformace automobilů, reálné chování postav, skutečně vyhlížející přírodní úkazy, což vedlo k dramatickému zlepšení simulace fyziky ve hrách.



Obrázek 2.4: Hra Crysis, která podle světových magazínů patří mezi technologické špičky roku 2007

Obrázek 2.5: Porovnání prvního dílu Command & Conquer z roku 1995 (vlevo) a nejnovější díl Command & Conquer 4 z roku 2010 (vpravo)[4]

[5] Anglicky General-Purpose computing on Graphics Processing Units. Jedná se o náročné obecné výpočty prováděné na grafických kartách, které jinak vypočítává procesor díky programovatelné části GPU.

3 Počítačová grafika

Počítačová grafika (*Computer graphics*) je součástí oboru informatiky, která se zabývá analýzou (*interpretací*) nebo tvorbou (*syntézou, generováním*) grafické obrazové informace[9]. Jednou z možných rozdělení počítačové grafiky je podle počtu zpracovávaných dimenzí prostoru na 2D nebo 3D.

3.1 Historie počítačové grafiky

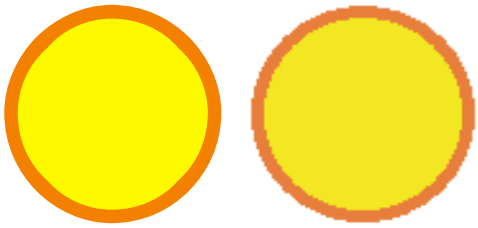
Historie počítačové grafiky začíná mnohem dříve, než byly počítače uvedeny v provozu. Významný podíl na vzniku mají vynálezci, speciálně matematici, bez jejichž objevů by grafika nemohla existovat. Mezi první matematiky patřil Euklidus, který ke konci 4. stolení př. n. l. vydává dílo o základech matematické geometrie. Na Euklida později navazuje řada matematiků z dob renesance, kteří se snaží rozvinout jeho základní myšlenku o matematické geometrii. V 19. století byl významným objevem maticový zápis od Jamese Sylvestera, bez čeho by byly transformace ve 3D značně komplikované. V roce 1950 Ben Lapovsky začíná vytvářet první elektronické obrázky za pomoci osciloskopu, kde zaznamenává paprsky elektronů na film. Pojem počítačová grafika zavádí až William Fetter, designér společnosti Boeing, roku 1960. O 4 roky později také vytváří první model lidské postavy pro studii kabiny letadla[5]. V letech 1968 – 1970 se vědci zaměřují na 3D scény, díky čemuž byl vynalezen ray-tracing. Henri Gouraud a Bui Tuong Phong objevují způsoby stínování a principy výpočtů odlesků, rovněž používané i dnes. Grafické programy nebo aplikace začínají vznikat v 80. letech 20. století. Mezi první patří konstrukční program AutoCAD, za kterým stojí Dan Drake a John Walkner. Další programy na sebe nenechaly dlouho čekat. Roku 1984 se objevuje první 3D grafický software a to Polhemus, jen o pět let později spatřil světlo světa úspěšný vektorový editor Corel Draw[5][6]. Důležitým milníkem byl také vznik grafických knihoven OpenGL (*viz. 4.2*) a DirectX (*viz. 4.1*).

Srovná-li se vývoj počítačových her a počítačové grafiky, může se zdát, že hry byly opožděné, jelikož už v roce 1972 byly známy základy Phnogova stínování a na herní scéně se teprve objevila jednoduchá hra Pong. Vše bylo způsobené tím, že obor počítačová grafika patřila pouze na vědeckou a akademickou půdu. Až díky rozvoji počítačů do domácností se počítačová grafika, jakožto hlavní zpětná vazba mezi člověkem a počítačem, přesouvá mezi spotřebitele.

3.2 Rastrová grafika

Rastrová grafika je způsob popisu (*uložení, definice*) zpracované a zobrazené informace nejčastěji ve formě 2D matice, diskrétně. Tento popis dat získáme buď manuálně, nebo syntézou (*generováním nebo převodem z jiného popisu*), případně snímáním (*kamerou, fotoaparátem*). Jeden prvek matice se nazývá pixel, který nese informace například o barvě a jasu bodu[9]. Existuje ale i voxel[6], texel[7]. Obraz je složen pouze z bodů. Rozlišení rastrové matice vůči zobrazeným objektům je dopředu dané a nelze je jednoduše měnit bez snížení kvality zobrazení. Jediná možnost změny rozlišení je převzorkování.

Současná zobrazovací zařízení jsou založena na rastrovém principu (*monitory, projektory*), ale pro uložení v paměti počítače, zpracování a syntézu je vhodnější vektorový popis. Proto se při zobrazení vektorových dat provádí rasterizace (*proces převodu matematických primitiv na jejich odpovídající zobrazení ve formě rastrové grafiky*). Rasterizace je plně v režii grafických knihoven, zatímco vektorizace je jako automatický proces nespolehlivá. Proto vyžaduje zásah tvůrčí práce člověka[9].



Obrázek 3.1: Ukázka rozdílu mezi vektorovou a rastrovou reprezentací

3.3 Vektorová grafika

Vektorová grafika je způsob popisu (*uložení, definice*) zpracovávané a zobrazované informace ve formě skupiny matematických primitiv (*úsečky, kružnice, křivky, polygony*), analyticky. Tento popis dat získáme buď, manuálně nebo syntézou (*generováním nebo převodem z jiného popisu*) [9]. Při zobrazování se objekty převádí na rastrové zobrazení. Jednotlivé objekty (*entity*) jsou definovány pomocí analytické matematiky a parametrů, takže nejsou závislé na rozlišení, díky tomu lze měnit velikost (*zmenšování, zvětšování*) obrazu bez ztráty kvality a pracovat s každým objektem samostatně[9].

[6] Částice objemu, představující hodnotu v pravidelné mřížce třídimenzionálního prostoru počítačové grafiky. Je to analogie k pixelu, který reprezentuje 2D grafiku.
[7] Jedná se o základní jednotku textury.

4 Grafické knihovny

Vývojář počítačových her má v současné době na výběr ze dvou knihoven. Mezi ně patří DirectX vyvíjen a uváděn na trh společnosti Microsoft. Na druhé straně stojí OpenGL, jejichž značku vlastní společnost Silicon Graphics Inc. Obě knihovny se dynamicky a aktivně vyvíjí, aby podporovaly nové grafické technologie a usnadňovaly práci vývojářům. Díky tomu lze používat DirectX už ve verzi 11.0 a OpenGL 4.0.

OpenGL vždy nacházelo více využití v profesionální sféře (*CAD programy, vědeckotechnické vizualizace*). Důvod pro převahu OpenGL v profesionálním segmentu je částečně historický. Profesionální aplikace byly napsány pro IrixGL (*viz. 4.2.1*) a provozovány na pracovních stanicích SGI a následně předělány na OpenGL. Navíc, část profesionálních karet podporovala pouze OpenGL. Rovněž knihovna OpenGL disponuje škálou vlastností, které i když jsou bezvýznamné pro herní trh, jsou hojně využívány v profesionálních aplikacích[8][10].

DirectX naproti tomu nebyl vytvořen se záměrem práce v profesionální grafice a zaměřil se na herní segment a vývoj her.

4.1 DirectX

4.1.1 Vznik a historie

Vznik DirectX je spojen s operačním systémem Windows 95. Snaha Microsoftu byla, co nejvíce, rozšířit mezi uživatele nový operační systém, což znamená tvořit a vyvíjet spoustu kvalitních aplikací. Oblast tvorby počítačových her pod Windows 95 byla problematická, jelikož poskytované funkce byly pomalé a jejich možnosti omezené. Hlavní problémem byl především chráněný paměťový režim, který blokoval přístup ke grafické nebo zvukové kartě. Oproti tomu v systému MS-DOS tento přístup byl možný, a proto i po uvedení Windows 95 byly počítačové hry stále vyvíjeny pod MS-DOS[7][8].

Microsoft chtěl problém odstranit, což mělo za následek v roce 1995 vznik první verze DirectX pod názvem Windows Games SDK. Vývojáři navrhli grafickou technologii, která měla už od počátku maximální výkonnost. S postupem času, kdy narůstaly možnosti grafických karet a rychlost počítačů, bylo nutné provádět časté aktualizace a rozšiřovat možnosti DirectX s ohledem na kompatibilitu se staršími verzemi. V praxi to znamená, že pokud je na stanici nainstalována verze DirectX 8.0, lze zde bez problémů spustit i aplikaci vyvíjenou pod verzi DirectX 6.0. Problém nastává

až ve verzi DirectX 10.0, která má kompletně přepracované rozhraní a starší verze DirectX (*od verze 6.0 a níž*) jsou zde pouze emulovány, což má za následek pokles výkonu[7].

4.1.2 Klady a zápory

Předností DirectX je přítomnost modulů potřebné pro vývoj počítačových her. Jako je modul pro práci se zvuky, periferiemi, síťovým rozhraním. Práce s jednotlivými moduly je podobná. Další výhodou je, že celá knihovna je pravidelně aktualizována a jsou zde stále přidávané nové funkce, u nichž není podstatné, jestli je grafická karta podporuje, jelikož se dají bez menších požadavků aplikovat softwarově. U složitých her a aplikací je výhodou objektová orientace celého rozhraní. Mezi záporné body DirectX patří podpora pouze pro systém Microsoft Windows[7].

DirectDraw	Práce s rastrovou 2D grafikou pro maximální výkonnost. Od verze DirectX 8 spojeno s Direct3D.
Direct3D	Rozhraní určené pro práci s 3D grafikou, kde je kladen důraz na rychlost a výkonnost s podporou hardwaru.
DirectX Input	Rozhraní poskytující služby nejrůznějších vstupních zařízení (*klávesnice*, *myš*, *gamepad*).
DirectX Audio	Podpora komunikace se zvukovými kartami. Možnost vytvářet aplikace se zvuky a hudbou. Podpora 3D zvuku.
DirectMusic	Podpora zvuku na vyšší úrovni než DirectSound. Možnost přehrávání více zvukových formátů. Práce s dynamickými zvuky.
DirectPlay	Podpora síťové komunikace ve hrách. Lze komunikovat jak pomocí PTP, tak i systémem klient/server.

Obrázek 4.1: Základní popis rozhraní DirectX

4.2 OpenGL

4.2.1 Vznik a historie

Knihovna OpenGL vznikla v roce 1992 u společnosti Silicon Graphics Inc. (*SGI*). Jejímu uvedení předcházely výzkumy SGI v oblasti grafického modelování a realistické syntézy obrazu. V roce 1982 na Stanfordské univerzitě se zrodil koncept grafického počítače, který na principu proudového zpracování obrazových elementů vytvářel trojrozměrnou scénu. Na stejném základě se později u Silicon Graphics vyvíjely grafické stanice a knihovna IRIS GL, představující programátorské rozhraní právě pro podporu pipeline-renderingu. IRIS GL lze považovat za přímého předka OpenGL.

Obě tyto grafické knihovny mají mnoho společného. IRIS GL je souborem elementárních příkazů pro práci s 3D grafikou, které znamenají pouze volání služeb konkrétních grafických zařízení. Naproti tomu OpenGL knihovna byla od počátku koncipována jako hardwarově nezávislá. V průběhu času se IRIS GL zbavovala problematických rysů a dá se říci, že poslední verze IRIS GL je s OpenGL téměř kompatibilní[8][10].

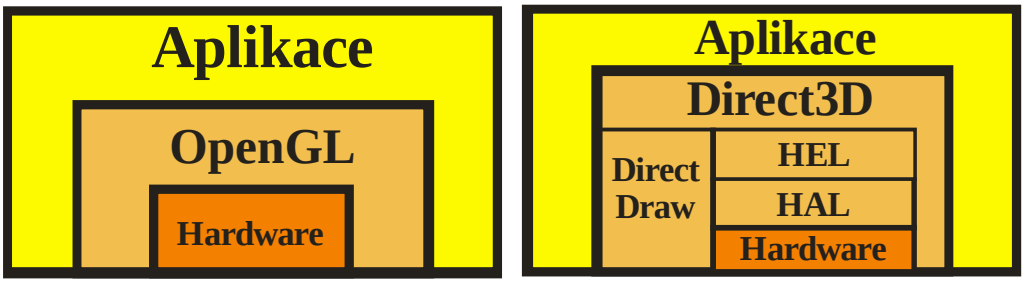
4.2.2 Klady a zápory

Jeden z hlavních pilířů OpenGL je přenositelnost a nezávislost na operačním systému a hardwaru. Aplikace implementovány pomocí knihovny OpenGL mohou být bez větších problému přeloženy a spustitelné jak pod systémem Windows, Linux, MacOS a dalších.

Větší úskalí mohou nastat při vývoji počítačových her, protože OpenGL je vykreslovací knihovna. Ostatní elementy aplikace jako je audio, obsluha oken, síťové rozhraní, práce se soubory, musí být implementovány samostatně. K tomu lze využít například knihovnu SDL a její podknihovny (SDL_net, SDL_image, SDL_mixer), případně knihovnu FMOD pro práci s audio prvky. To sebou nese nutnost učit se práci s dalšími knihovnami navíc.

4.2.3 Struktura OpenGL a Direct3D

Modul Direct3D, součást knihovny DirectX, se věnuje pouze vykreslování a byl vyvinut pro jednotný přístup k různým grafickým čipům a aby usnadnil implementaci počítačových her pod systémem Windows. Zároveň se stal přímým konkurentem zaběhlého rozhraní OpenGL. Direct3D a OpenGL mají za úkol abstrahovat programátora od konkrétního hardwaru.



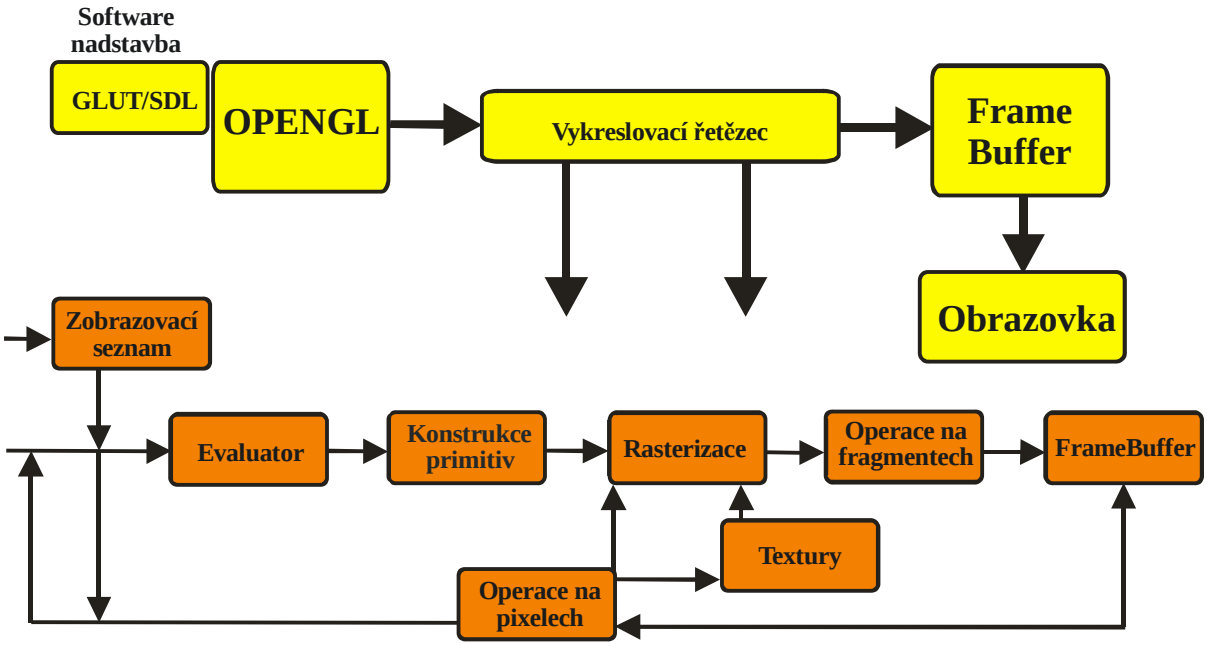
Obrázek 4.2: Bloková struktura OpenGL a Direct3D. Od verze DirectX 8 je DirectDraw součástí Direct3D[8]

OpenGL tedy abstrahuje hardware a volá jeho služby. Závisí pouze na programátorovi, zda službu vykoná procesor nebo grafická karta a díky tomu se každá implementace z hlediska aplikace chová stejně. Samotný Direct3D je oddělen od hardwaru, protože předem nelze určit, jaké funkce bude grafická karta přímo podporovat. Každá funkce je nejprve filtrována vrstvou HEL (*Hardware Emulation Layer*), která rozhodne, zda jí grafická karta umí provést. Pokud ji nelze provést, funkce je

vypočítána softwarově pomocí této vrstvy. Jestliže grafická karta funkci podporuje, použije se vrstva HAL (*Hardware Activation Layer*), která provede volání na konkrétní hardware[8].

4.2.4 Princip práce OpenGL

Z programátorského hlediska se OpenGL chová jako stavový automat, jehož úkolem je zobrazit obraz pomocí grafické karty. Během zadávání příkazů pro vykreslování lze průběžně měnit vlastnosti vykreslovaných primitiv (*barva*, *průhlednost*) nebo celé scény (*volba způsobu vykreslování*, *transformace*) a toto nastavení zůstane zachováno do té doby, než se explicitně změní. Výhoda stavového automatu spočívá především v tom, že funkce pro vykreslování mají menší počet parametrů a pouhým jedním příkazem lze globálně změnit způsob vykreslení celé scény. Mnoho stavových proměnných odkazuje na módy, které jsou povoleny či zakázány příkazy `glEnable()` nebo `glDisable()`. Vykreslování scény se provádí procedurálně. Voláním funkcí OpenGL se vykreslí výsledný rastrový obraz do framebufferu. OpenGL umožňuje nastavit režim práce pomocí dvou módů. Každý mód se dá měnit nezávisle. Nastavení jednoho neovlivňuje nastavení druhého módu. Zpracování elementárních objektů může probíhat buď v módu bezprostředním (*immediate mode*) nebo pomocí zobrazovacího seznamu (*display-list mode*). Bezprostřední mód spočívá v tom, že všechny akce s primitivy se provádějí přímo v ten okamžik, kdy jsou programem zavolány. Tento přístup je vhodný pro statické scény, kde se objekty neopakují. V případě, že má být vytvořena v OpenGL animace, může být tento přístup neefektivní. V každém snímku, i když se od předchozího liší jen minimálně, by musely být znovu specifikovány všechny objekty na scéně. Z toho důvodu je do OpenGL zařazen i mód zobrazovacího seznamu (*display-list mode*), který využívá pomocnou datovou strukturu. V té se uchovávají předdefinované objekty a na místě volání se potom vyvolávají pouze odkazy na tyto objekty[8][10][12].



Obrázek 4.3: Vykreslovací schéma OpenGL[8][10]

OpenGL posílá do vykreslovacího řetězce příkazy, buď přes zobrazovací seznam, nebo přímo přes linku proudového zpracování. V první fázi zpracování se případně složitější objekty (*objekty z knihovny GLU*) převádí na jednodušší, se kterými umí OpenGL přímo pracovat. Další fáze zpracuje body, úsečky, polygony a případná osvětlení. Je prováděno ořezávání na rozměr projekčního okna. Rasterizér vytváří sérii adres do framebufferu, při použití dvojrozměrného popisu čar, úseček a polygonů. Nyní každý bod získá své vlastnosti (*barva, světelný zdroj*). Ve fázi rasterizace se nanáší textury na požadovaný objekty. Výsledek rasterizéru není hotový obraz, ale množina fragmentů. Jednomu fragmentu odpovídá jeden pixel výsledného obrázku, který nese navíc informace o barvě, světle, hloubce, popřípadě obsahuje souřadnice textury. Fragmenty lze zpracovávat a docílit tím různých efektů (*vyhlazení hran, míchání barev*). Takto výsledný obraz je uchován do framebufferu, kde proudové zpracování končí a výsledek se zobrazí na obrazovce[8][12].

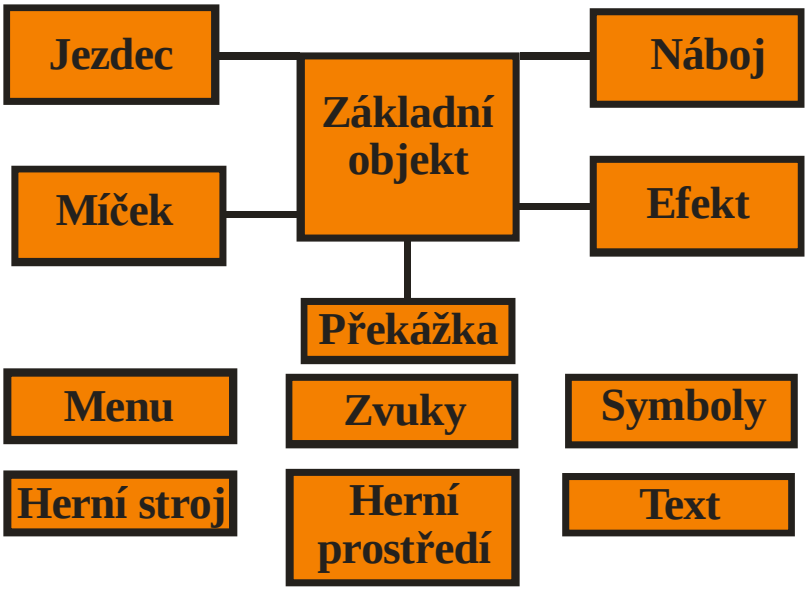
5 Návrh

5.1 Programové řešení

Pro tvorbu aplikace bude zvolena grafická knihovna OpenGL z důvodu přenositelnosti zdrojových textů. Správu oken, zvuků a ovládání zajistí knihovna SDL, umožňující rovněž plnou přenositelnost na ostatní operační systémy. Aplikace je naimplementována pomocí programovacího jazyka C++ z důvodu jeho rychlosti a bezproblémové práci s knihovnami OpenGL a SDL. Díky zvoleným knihovnám je celá aplikace a zdrojové texty plně kompatibilní se systémem Windows a Linux. Jako vývojové prostředí slouží Microsoft Visual Studio 2008.

5.1.1 Objektový návrh

Aplikace je naimplementována objektově-orientovaným přístup, jelikož umožňuje dědičnost, zapouzdření a polymorfismus. Na přiloženém obrázku lze vidět základní náčrt objektového návrhu. Kompletní diagram tříd je obsažen v příloze 1.



Obrázek 5.1: Náčrt objektového návrhu

5.1.2 Základní grafické elementy OpenGL

Pomocí příkazů OpenGL lze vykreslovat pouze základní geometrické elementy (*primitiva*), které jsou základním stavebním kamenem celé aplikace. Z těchto elementů se potom skládají složitější tělesa a celá scéna. Základními elementy jsou v OpenGL samotný bod, úsečka, polygon, bitmapa[8] a pixmapa[9].

Při vykreslování každého grafického elementu musí být zadán první příkaz `glBegin(typ přepínače)`, kterým knihovna OpenGL dostane informaci o začátku zadávání jednotlivých vrcholů. Vykreslování se ukončí párovým příkazem `glEnd()`. Mezi tyto příkazy lze zadávat libovolný počet vrcholů primitiv pomocí `glVertex*()`. Pro 2D scénu jsou zadány pouze souřadnice (x, y) a vrcholy se zapisují příkazem `glVertex2*()`[1][10].

5.1.2.1 Přehled používaných základních grafických elementů

Nejčastějším používaným primitivem je bod a úsečka. Úsečka je popsána dvěma koncovými body (*vrcholy*). Zadávání úsečky začíná příkazem `glBegin(GL_LINES)` a končí `glEnd()`. Úsečka začínající v bodě [0;0] a končící v [10;10] je vykreslena s pomocí funkcí OpenGL následovně.

```
glBegin(GL_LINES);
      glVertex2f(0.0f,0.0f);
      glVertex2f(10.0f,10.0f);
glEnd();
```

Obrázek 5.2: Vykreslení přímky pomocí OpenGL

Pro zadání řetězce úseček stačí změnit pouze typ přepínače na `GL_LINE_STRIP` a všechny zadané vrcholy budou následně spojeny. Případně pro vytvoření smyčky z úseček je používán přepínač `GL_LINE_LOOP`[1].

V aplikaci jsou využívány základní vlastnosti elementů jako je změna barvy pomocí příkazu `glColor4*(Red, Green, Blue, Alfa)`. Rovněž lze také nastavovat šířku čar příkazem `glLineWidth(GLfloat sirka)`, implicitně je šířka nastavena na hodnotu 1 (*odpovídá 1 pixelu na obrazovce*). U polygonu se může nastavit typ vykreslování, buď pouze vrcholů (`GL_POINT`), hran (`GL_LINE`) nebo výplně (`GL_FILL`) pomocí příkazu `glPolygonMode(GLenum face, GLenum mode)`. Poslední důležitou vlastností je používání antilasingu primitiv (*vyhlazovaní, rozmazání hran*). Základní antilasing se zapíná příkazem `glEnable(GL_POINT_SMOOTH)` u bodu, `glEnable(GL_LINE_SMOOTH)` u úseček a glEnable`(GL_POLYGON_SMOOTH)` u

[8] Jednobarevný rastrový obrázek.
[9] Vícebarevný rastrový obrázek.

polygonu. Vypnutí se provede analogicky příkazem `glDisable(mode)`. Za znak * se dosadí příznak datového typu (`i-GLint,f-GLfloat, d-GLdouble`) parametrů[1][10].

5.1.3 SDL

Knihovna SDL (*Simple DirectMedia Layer*) byla navržena jako obecné, nízkoúrovňové aplikační programové rozhraní pro tvorbu her a obecně multimediálních aplikací. Plně zastřešuje funkce operačních systémů a díky tomu umožňuje téměř stoprocentní přenositelnost zdrojového kódu. Nabízí možnost přehrávání hudby a zvuků, obsluhu vstupních zařízení jako je klávesnice, myš, CD mechanika a správa systému oken a událostí. Podporuje také více-vláknové programování. Obsahuje mnoho dalších podknihoven (*nadstaveb*) pro lepší práci s obrázky, zvukem, síťovým rozhraním a fonty[11].

- SDL_image: podpora více formátů obrázku
- SDL_sound a SDL_mixer: rozšířená práce se zvuky a hudbou
- SDL_ttf: podpora vykreslování fontů TrueType
- SDL_rtf: práce se RTF (Rich Text Format) soubory
- SDL_net: základní rozhraní pro síťovou komunikaci

SDL je napsáno v jazyce C, ale díky tzv. language bindings[10] je možné knihovnu použít v ostatních jazycích (*Java, Python, C++, C# a další*). Knihovna je uvolněna pod licenci GNU LGPL[11]

Knihovnu SDL jsem si vybral, jelikož má základní rozhraní pro práci s okny a událostmi, obsahuje licenci LGPL a umožňuje plnou přenositelnost zdrojového kódu na ostatní operační systémy. Z dalších knihoven jsem využil pro aplikaci pouze SDL_mixer pro práci s hudbou a zvuky.

5.2 Struktura hry

Prvním krokem při přenesení hry z myšlenky do skutečného světa je jasná představa, jak má hra vypadat. Důležitým faktorem je stanovit základní pravidla o rámcové podobě výsledné hry. Klíčovými oblastmi, kterými se budu zabývat, je základní koncept, příběh, grafika, ozvučení, ovládání a herní režimy[2].

5.2.1 Základní koncept

Po zhodnocení arkádových her (*viz. 2.2*), jsem se rozhodl k mému obrazu přepracovat starou a známou hru z roku 1986 od společnosti Taito – Arkanoid. Existuje mnoho adaptací této hry, i přes to jsem se ale snažil aplikaci pojmout inovativním způsobem.

[10] Možnost knihovnu používat v mnoha ostatních programovacích jazycích

Základním kamenem hry je ovladatelný jezdec hráčem, jehož úkolem je zachytávat vystřelený míček a odrazem zničit přichystané překážky. Překážky mají postupem času odlišné vlastnosti. Z tohoto důvodu má jezdec za určitý čas možnost vypustit speciální střely, které pomáhají při ničení překážek.

5.2.2 Příběh

U podobných her se příběh, nebo jiné vyprávění děje vyskytuje málo. Proto hra obsahuje poutavou a zajímavou dějovou linii. Samotný děj hry se odehrává v 80. letech 20. stolení, symbolicky přesně v roce 1986, kdy vyšla původní hra (*viz. 2.1*). Dějová linie vychází z alternativní historie lidstva a je vedena formou rozhovoru, probíhající před každou úrovní. Hlavními postavami jste vy, jako hráč jménem James a vždy vás doprovází rádce (*mentor*) ve starším věku Mr.Trefford, jehož snahou je vychovávat ve vás bojového ducha s citem pro rozhodování.

Lidstvo se z velké krize v 30. letech nevzpamatovalo tak snadno. 2. světová válka ani neproběhla a celé povědomí lidstva se vyvinulo jinak. Nikdy se nepostavily účinné zbraně. Celý technologický vývoj směřoval odlišným směrem. Svět ovládla repulsní technologie, schopna po kontaktu udělit předmětu obrovskou rychlost a směr. Díky tomu vznikl sport Blastpoint. Lidé sedí přímo v repulsních plošinách, které ovládají a snaží se zachytit obrovskou kouli. Díky repulsní technologii vystřeli kouli zpět do předem připravených překážek, které jsou v herní aréně předem poskládané do různých tvarů. Vy jako mladý, nadějný hráč se chcete stát světovým šampiónem v tomhle sportu a usilovně bojujete o titul, když v tom náhle… Jedná se první úryvek ze hry, který přibližuje dění a základní příběh.

5.2.3 Grafika

Grafická stránka patří mezi důležitou část hry. Zobrazení je počítáno pomocí 3D akcelerace, ale je omezeno na 2D zobrazení, aby se zachovala dobová atmosféra starých her. Jelikož v počátcích hry (*viz. 2.1*) neměly žádné textury[11], případně neobsahovaly sprity[12], z toho důvodu je aplikace naimplementována bez těchto prvků. Celá hra je tudíž vykreslena pouze pomocí základních geometrických primitiv, jako je bod, úsečka a polygon. Díky těmto prvkům hra působí historicky a lehce zastarale. Všechny texty a písmena jsou rovněž vykresleny pomocí primitiv z důvodu, aby hra neobsahovala žádnou práci s texturou, případně s knihovnami, které se starají o práci s texty, jako je SDL_ttf.

[11] Textura je obrázek, kterým je obalené vymodelované těleso. Proces nanášení textur na objekt se nazývá texturování.
[12] Jedná se o dvourozměrný obrázek, který definuje určitou animační fázi požadovaného objektu

Aby byla zaznamenána historická podoba hry v maximální míře, aplikace obsahuje základní barvy z barevných modelů RGB a CMY v kombinaci s bílou. Všechny barevné složky jsou podpořeny také alfa-kanálem, pro obohacení grafického výstupu.

5.2.4 Ozvučení

Většina počítačových her obsahuje zvukové stopy, případně melodie, které dodávají hře správnou atmosféru. Proto je hudba a zvuk vytvořen v takovém rozsahu, aby odpovídal tématice a prostředí hry. Pro tyhle účely využívám počítač Apple s operačním systém MacOS, který má zajímavé nástroje pro tvorbu hudby jako je Logic Studio.

5.2.5 Ovládání

Aplikace pro ovládání používá pouze funkce jedné periferie a to klávesnice. Tomu je přizpůsoben celý systém uživatelského rozhraní, aby byl pro uživatele maximálně přehledný a svižný. Každá akce v menu a ve hře se vyvolává příslušnou klávesou (*písmenem, číslem*). V přiložené tabulce je navrženy systém ovládání.

Klávesa	**Akce**	**Klávesa**	**Akce**
V hlavním menu		**Ve hře**	
H	Začátek hraní příběhové linie	**M**	Návrat do menu
P	Seznámení s příběhem	**U**	Uložení stávající úrovně
O	Popis základního ovládání	**S**	Vypnout/zapnout hudbu a zvuky
A	Informace o autorovi	**Num +**	Zvýšení hlasitosti
K	Konec hry	**Num -**	Snížení hlasitosti
L	Nastavení lehké obtížnosti	**Š. nahoru**	Vystřelení střely
T	Nastavení těžké obtížnosti	**Š. doleva**	Pohyb jezdce doleva
Z	Zpět do rozehrané hry	**Š. doprava**	Pohyb jezdce doprava
N	Nahrát uloženou pozici	**Mezerník**	Pokračovat do další úrovně
1 až 9	Hrát libovolnou úroveň		
S	Vypnout/zapnout hudbu a zvuky		

Obrázek 5.3: Návrh a popis ovládání

5.2.6 Herní režimy

Pro zvýšení zábavnosti má aplikace dva herní režimy. První režim se stará o příběhovou linii obsahující devět rozličných úrovní. V příběhové linii má hráč možnost si hru i kdykoliv uložit a

později v dané části pokračovat. Po dohrání příběhové linie se hráči odemknou (*otevřou*) všechny dostupné úrovně. Dále pro zvýšení zábavnosti hra obsahuje dva režimy obtížnosti.

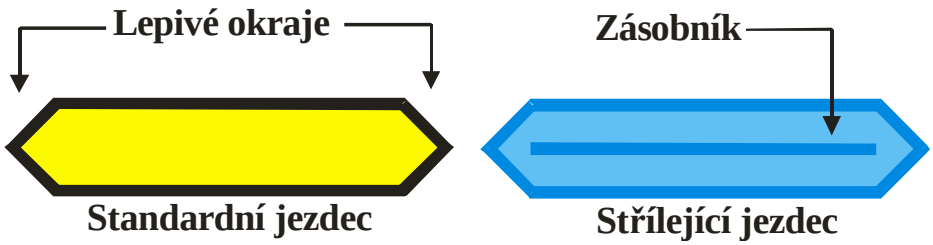
5.3 Herní systém

Princip aplikace vychází z původní hry Arkanoid. Úkolem hráče je pomocí jezdce zachytávat a odrážet míček, aby zůstal na hrací ploše. Odražený míček při dotyku ničí přichystané překážky ve hře. Jakmile se hráči povede zničit všechny překážky, postupuje do další úrovně.

Hlavní jednotkou hry je herní stroj (*angl. game engine*), který se stará o celou správu aplikace, od inicializace hry, vykreslování jednotlivých úrovní, správa hudby a ovládání až po hlavní smyčku celého programu. Každá úroveň obsahuje vždy několik překážek, počet se pohybuje od 40 až po 100 v závislosti na aktuální úrovni a zvolené obtížnosti. Po dohrané úrovni je vytvořen součet skóre.

5.3.1 Jezdec

Patří mezi hlavní pilíře hry, jeho úkolem je správně zachytit míček, aby nespadl mimo herní plochu, a zároveň musí zničit předem připravené překážky. Jezdec má několik základních vlastností, které mu dopomáhají dostat se do další úrovně, jako jsou stále lepivé okraje, kde se míček jednoduše přichytí *(tato akce je ve hře penalizována -2 body za každé přilepení*) tak i možnost střílet. Aby mohl hráč využít vlastnosti střílení, musí být zručný a pozorný, jelikož jezdec se přemění pouze tehdy, pokud hráč úspěšně 10x za sebou odrazí míček, aniž by mu spadl mimo herní plochu. Poté jezdec získá na krátký časový interval zbraň, která dokáže zničit překážky v úrovni. Pokud chce hráč vystřelit, musí počkat, než ukazatel zásobníku bude zcela naplněn, z toho vyplývá, že jezdec má také vlastní nabíjející systém, který pracuje zcela automatizovaně.

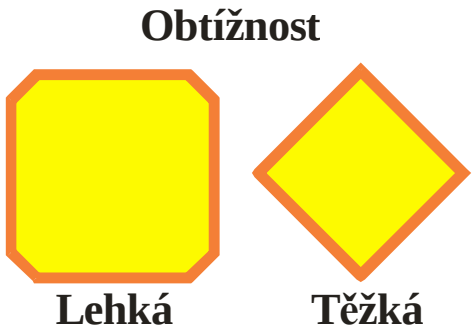


Obrázek 5.4: Vzhled a vlastnosti jezdce

5.3.2 Překážka

Ústřední nepřítel ve hře. Každá úroveň obsahuje rozdílné množství překážek, které postupem času získávají různé vlastnosti. Překážka má dva různé tvary, závisející na obtížnosti. Z toho vyplývá i rozdílné chování při odrazech a jejích počet v jednotlivých úrovních. Základní vlastnosti se dělí na tři

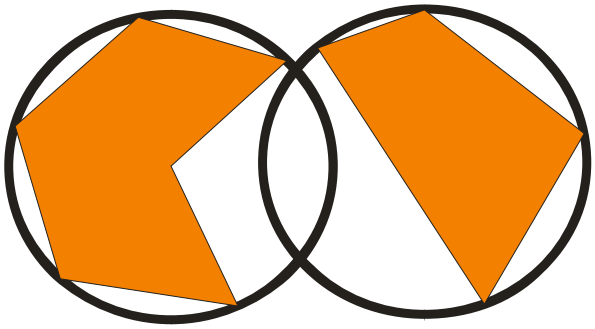
kategorie a každá vlastnost má svou barvu. Pokud je překážka zničitelná pouze jedním dotykem míčku (*získá hráč +10 bodů*), má základní barvy RGB modelu (*červená, zelená, modrá*), překážky zničitelné na dva doteky (*získá hráč +15 bodů*) jsou vybarveny základními barvami CMY a nezničitelné jsou bílé.



Obrázek 5.5: Vzhled překážky

5.4 Kolizní systém

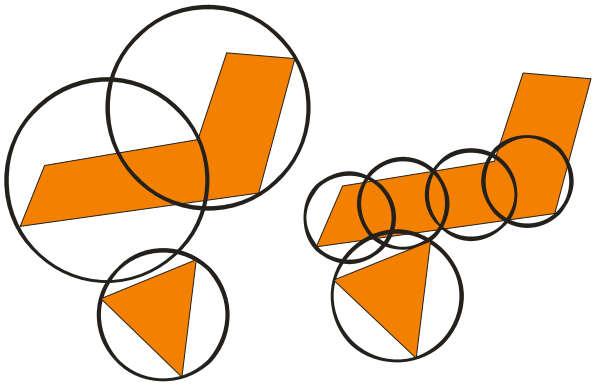
Detekce kolizí, pokud má být 100% funkční je náročná na strojový čas. Je proto důležité si předem dobře rozmyslet, u kterých objektů je jí třeba a u kterých ne. Zajímavým způsobem detekce kolizí je metoda průniku dvou sfér. Měří se zde vzdálenost středů dvou objektů. Pokud je obdržená hodnota menší než pevně daná hranice (*součet poloměrů dvou kružnic – uvažuji-li ve 2D*), oznámí funkce kolizi[13].



Obrázek 5.6: Metoda průniku dvou sfér

Na obrázku 5.6 je vidět nevýhodu popsané metody. Ta může totiž ohlásit kolizi i v případě, že se objekty vůbec nedotýkají. Na druhou stranu je metoda průniku dvou sfér rychlá[13].

Další metodou je sférické rozdělení objektu. Podobně jako u předchozího způsobu, kde se hodnotila vzdálenost středů, se zde testovaný objekt rozdělí na množinu sekcí. Vzdálenost se testuje od středu každé sekce.

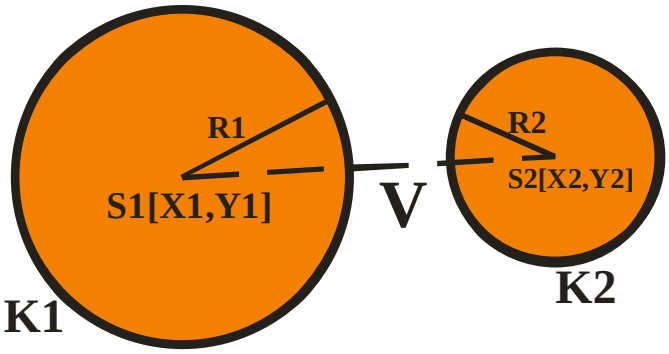


Obrázek 5.7: Sférické rozdělení objektu

Pokud se podaří najít vhodné rozdělení celého objektu, jedná se o další rychlou a efektivní metodu[13].

Ve 3D aplikacích se dají používat podobné metody pomocí koule, ale jsou nepřesné, proto vývojáři rádi používají metodu bounding box *(uzavřeny v krabici)*. Testovaný objekt je jakoby uzavřen do krabice, u které je testována kolizi. Jedná se o jednu z rychlých a efektivních metod. Další metoda, odvozenou od předchozí, je metoda kubického rozdělení objektu. Objekt se zde rozdělí na části tvořené krabicemi a kolize se testuje u každé z těchto částí. Přestože je tato detekce kolizí kvalitní, je náročná na implementaci.

Ve hře jsou při detekci kolizí použity výše vyjmenované metody, jak metoda průniku dvou sfér, tak sférické rozdělení objektů. Obě metody využívají základy analytické matematiky, přesněji vzájemnou polohu dvou kružnic a vzdálenost dvou bodů neboli velikost vektoru[13].



Obrázek 5.8: Poloha dvou kružnic

R_1 – poloměr kružnice K_1

S_1 – střed kružnice K_1 se souřadnicemi X_1,Y_1

R_2 – poloměr kružnice K_2

S_2 – střed kružnice K_2 se souřadnicemi X_2,Y_2

V – vektor

$$V = (X_2 - X_1; Y_2 - Y_1)$$

$$Velikost\ vektoru\ |V| = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

Poloha dvou kružnic:

$$R_1 - R_2 < R_1 + R_2 < |V|$$

Kružnice nemají žádny společný bod – nedošlo ke kolizi

$$R_1 - R_2 < R_1 + R_2 = |V|$$

Kružnice mají právě jeden společný bod – došlo ke kolizi

$$R_1 - R_2 < |V| < R_1 + R_2$$

Kružnice mají právě dva společné body – došlo ke kolizi

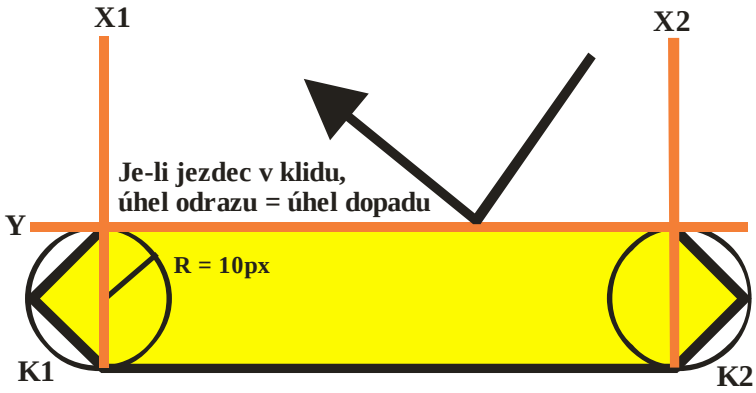
```
int vzdalenost = sqrt(pow(O1.X()-O2.X(),2) + pow(O1.Y()-O2.Y(),2));
if(vzdalenost <= (O1.R()+O2.R()))
{
      /*Nastala kolize – reakce na kolizi*/
}
```

Obrázek 5.9: Pseudokód pro počítání kolize dvou kružnic, který je použit v aplikaci
O1.X(), O1.Y() a O2.X(), O2.Y() – metody pro získání souřadnice x,y pro objekt 1 a 2
O1.R() a O2.R() –metody pro získání poloměru objektu 1 a objektu 2

5.4.1 Kolizní systém jezdce

U jezdce mohou v herním prostředí nastat pouze dva druhy kolizí. Kolize s herním prostředím, nebo s míčkem. Při kolizi s herním prostředím využívám pouze vlastnosti osy X, po které se jezdec pohybuje. Pokud pozice, kterou získám pomocí metody `GetPosition_X()` je za hranou herního prostředí, automaticky se nastaví pozice díky metodě `SetPosition_X(GLfloat x)` tak, aby jezdec zůstala v herním prostředí.

Druhý typ kolize je s míčkem. Zde jsem využil předchozí znalosti o kolizních systémech a implementoval jsem metodu průniku dvou sfér, která je popsána v kapitole 5.4, s kombinací porovnávání souřadnic. Díky těmto principů se odrazy míčku od jezdce vypočítávají na základně fyzikálních zákonu.

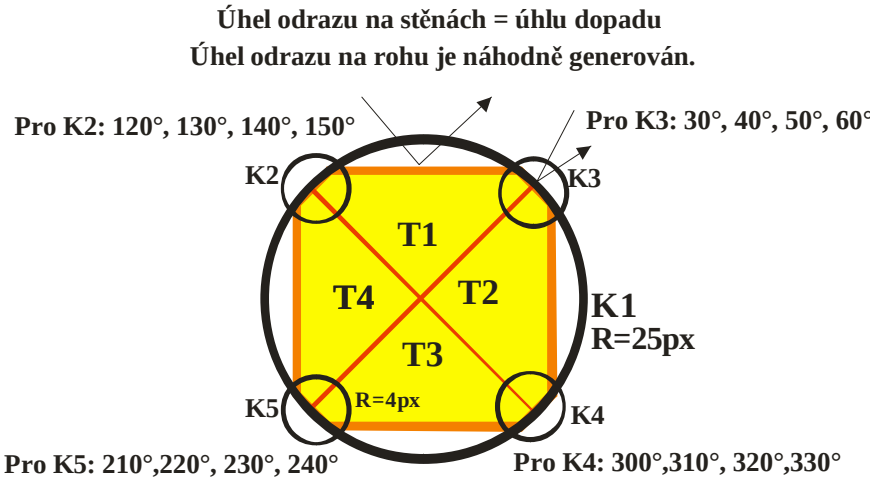


Obrázek 5.10: Ukázka kolizního systému jezdce

Jestliže nastane kolize s kružnicemi K1 nebo K2, míček se automaticky přilepí a zůstane na hraně jezdce, dokud ho hráč opět nepošle do hry. V takovém případě je úhel směru míčku pro K1 nastaven na 100° respektive na 80° pro K2. Hlavní kolize s jezdcem nastane tehdy, pokud se y-ová souřadnice míčku dostane pod úroveň bodu Y a zároveň se x-ové souřadnice míčku bude nacházet v prostoru mezi body X1 a X2. V takovém případně je úhel odrazu míčku roven úhlu dopadu, je-li jezdec v klidu. Jestliže se jezdec pohybuje směrem doleva, tak od výsledného úhlu odrazu se odečte určitá hodnota, přesněji 20°. Pohybuje-li se jezdec doprava, tak k výslednému úhlu odrazu se naopak připočte 20°.

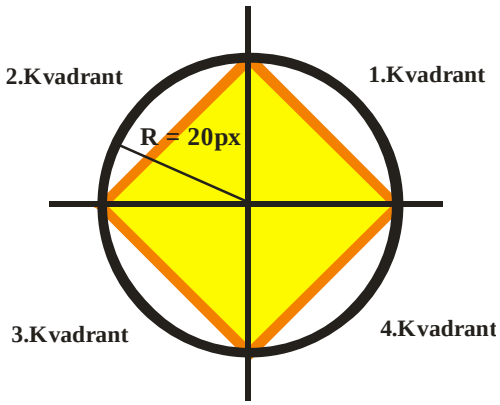
5.4.2 Kolizní systém překážky

Ve hře se vyskytují dva druhy překážek závislé na zvolené obtížnosti (*viz. 5.3.2*). Pro každou obtížnost je kolizní systém vypočítáván jiným stylem. Zde plně využívám metodu sférického rozdělení objektu.



Obrázek 5.11: Sférické rozdělní překážky pro lehkou obtížnost

Pokud nastane kolize míčku s kružnicí K1, musím zaznamenat, v jakém místě k tomu došlo, abych mohl vypočítat přesný úhel odrazu, který se v téhle situaci *(odrazy na stěnách)* bude rovnat úhlu dopadu dle fyzikálních zákonů. Přesnou pozici kolize zjistím pomocí analytické geometrie a vzájemné polohy bodu *(jedná se o bod na kružnici míčku)* a trojúhelníku *(T1, T2, T3 a T4)*. K tomu mi pomohou obecné rovnice obou úhlopříček. Kvůli zkoseným rohům překážky bylo nutné zjistit, zda dojde ke kolizi s rohem. Tento problém mně pomohly vyřešit další 4 kružnice *(K2, K3, K4, K5)*, které se nachází na hraně překážky. Nastane-li kolize s některou z těchto kružnic, úhel odrazu pro míček je nastavován podle obrázku 5.11 pomocí náhodného generátoru čísel `rand()`. Podobný systém počítání kolizi jsem implementoval i pro těžkou úroveň, s tím rozdílem, že výpočet je mnohonásobně rychlejší, jelikož stačí zjišťovat kolizi s jednou kružnicí, která obaluje celý objekt, jak je vyzobrazneno na obrázku 5.12.



Obrázek 5.12: Sférické rozdělení překážky pro těžkou obtížnost

Úhly odrazu míčku jsou řešeny stejným stylem, jako u lehké obtížnosti. Nastane-li kolize, zjistí se, v jakém kvadrantu leží nyní střed míčku podle analytické geometrie a změní se směr. Je-li kolize v 1. kvadrantu je úhel opět náhodně generován pomocí funkce `rand ()` v rozsahu 30°, 40°, 50° nebo 60°. Ostatní kvadranty generují stejné úhly podle obrázku 5.11.

5.5 Geometrické transformace

Díky použité vektorové grafice, jsou objekty popsány pomocí vertexu neboli uzlových bodů. Při zobrazování, vytváření a upravování objektů je potřeba provádět jejich posunutí, otočení, zmenšení nebo zvetšení. Těmto operacím se pak souhrnné říká geometrické transformace. Geometrické transformace můžeme chápat jako změnu pozice vrcholu v aktuálním souřadném systému, nebo jako změnu souřadného systému. Díky správným transformacím lze vytvářet pohyby a animace v počítačové grafice. Transformace ve 2D kartézské soustavě *(ale i ve 3D*) používají transformační matice. Ve 2D počítačové grafice jsou používány matice o rozměru 3x3[9].

- **Posunutí**

Při posunutí objektu dochází ke změně vrcholů v souřadnicovém systému pomocí transformační matice posunutí, která je definována následovně:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix}$$

Pomocí knihovny OpenGL, lze tuhle transformační matici posunutí zapsat pomocí funkce[9]:

```
glTranslatef(dx, dy, 0.0f)
```

- **Otočení**

Otáčení objektu nebo bodu v rovině je vytvářeno pomocí úhlu. Ve 2D má smysl otáčet objekty pouze podle jedné osy, z-tové. Transformační matice otočení proti směru hodinových ručiček pomocí úhlu alfa, kde střed otáčení je v počátku souřadného systému, je definována následovně[9]:

$$M = \begin{bmatrix} cos\alpha & sin\alpha & 0 \\ -sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Knihovna OpenGL tento problém řeší pomocí funkce:

`glRotatef(`α`, 0.0f,0.0f,1.0f)`.

- **Změna měřítka**

Při změně měřítka dochází k vynásobení vrcholů faktorem změny S. Pokud je faktor větší než 1, odhází ke zvětšení, pokud je větší než 0 a menší než 1, dochází ke zmenšení[9].

$$M = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Díky knihovně OpenGL se změna měřítka neboli scale provádí pomocí funkce:

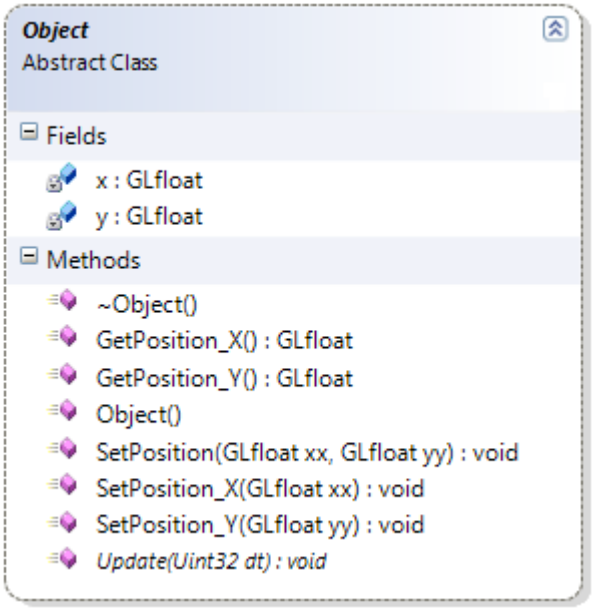
```
glScalef(Sx, Sy, 1.0f)
```

6 Implementace

Kapitola popisuje základní použité implementační prvky a jejich funkčnost. Celá aplikace je naprogramována pomocí jazyka C++ s využitím objektově orientovaného přístupu a obsahuje celkem dvanáct tříd. Pro urychlení komunikace objektů mezi sebou používám `inline` metody.

6.1 Třída Object

Základní třída abstraktní třída, ze které dědí funkčnost hlavní pohyblivé prvky hry. Obsahuje metody pro nastavování x-ové a y-ové pozice elementů *(jezdec, překážka, atd.)* ve scéně. Pro získání souřadnic jsou zde `inline` metody, jenž vrací pozici na x-ové a y-ové ose. Třída obsahuje jednu čistě virtuální metodu s názvem `virtual void Update(Uint32 dt) = 0`. Ta provádí časové aktualizace daného elemtů, jako je počítání pohybu na základně času.



Obrázek 6.1: Schéma třídy Object

6.2 Třída Rider

Jedná se o třídu, jejímž úkolem je funkčnost hlavního elementu hry, jezdce. Pohyb jezdce řeší metoda `void SetWay(GLfloat way)a GLfloat GetWay()`, kde při stisku šipky doprava, je předána metodě hodnota 1, při stisku šipky doleva hodnota -1, jinak hodnota 0. Pro zrychlení vykreslování jezdce na nových pozicích při pohybu je použit zobrazovací seznam. Jezdec je při inicializaci

vykreslen do display-listu *(zobrazovacího seznamu*), ze kterého je následovně pouze volán. Pohyb je řešen pomocí transformačních matic z kapitoly 5.5. Metody `void Draw()`, `void DrawLive()` mají za úkol vykreslovat jezdce a jeho životy.

6.3 Třída Circle

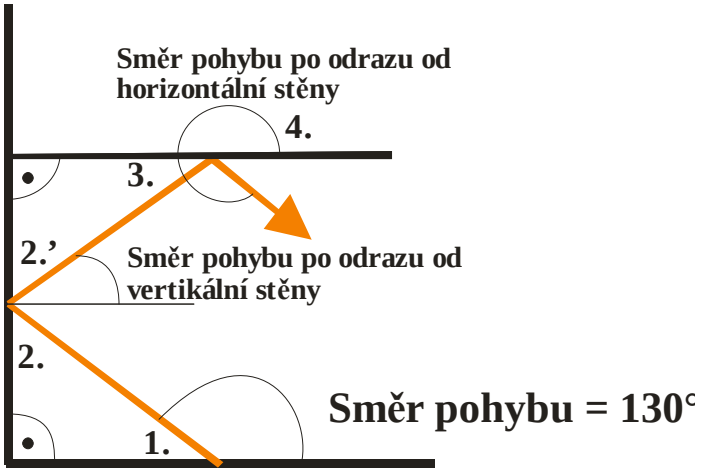
Hlavní činnost třídy je práce s míčkem. Počínaje reagování na pohyby, určování směru, až po samotné vykreslení. Vykreslení se děje pomocí metody `void Draw(GLfloat Red, GLfloat Green, GLfloat Blue)`. Kde parametry určují styl vybarvení elementu. Jelikož se jedná o kružnici a OpenGL neumí vykreslovat základní kružnice, proces vykreslování probíhá následovně:

```
glBegin(GL_LINE_LOOP);
     for(int i = 0; i < 360; i++){
          float theta = 2.0f*PI*float(i)/360.0f;
          GLfloat x = Radius*cosf(theta);
          GLfloat y = Radius*sinf(theta);
          glVertex2f(x, y);
     }
glEnd();
```

Obrázek 6.2: Vykreslení kružnice

Z důvodu náročnosti vykreslování je opět použit display-list (*zobrazovací seznam*), kde se při první inicializaci vykreslí celá základní kružnice.

Směr pohybnu míčku je určen úhlem (*celé desítky*) v rozmezí od 0° až po 360°. Pokud je míček přilepen k jezdci, je při výstřelu náhodně vygenerován úhel směru pohybu pomocí funkce `rand()` v rozmezí 10° - 170°. Metodou `void SetWay(GLfloat way)` se nastavuje požadovaný úhel pohybu. Metodou `inline GLfloat GetWay()`, získává aplikace nastavený směr míčku pro počítání změny pohybu při vyskytnutí kolize. Vznikne-li kolize se stěnou (*úhel odrazu se rovná úhlu dopadu*), záleži jestli je stěna vertikální, nebo horzintální.



Obrázek 6.3: Počítání úhlu pohybu

- **Kolize s vertikální stěnou:**

  Podle obrázku 6.3 lze zjisit podle pravidel o součtu úhlu v trojúhelníku, že úhel dopadu na levou stěnu, při směru pohybu 130° je 40° (*2.=2.'*). Směr pohybu po odrazu se pomocí metody nastaví `SetWay(GLfloat way)` nastaví na hodnotu 50°, aby byly zachovány fyzikální zákony o dopadech a odrazech. Starý úhel pohybu je získán pomocí metody `GetWay()`.

```
Novy_uhel_pohybu = 18.0f – Stary_uhel_pohybu; //aplikace
všechny úhly dělí hodnotou 10 pro snadnější manipulaci
```

Obrázek 6.4: Počítání úhlu pro odraz od vertikální stěny

- **Kolize s horizontální stěnou:**

  Opět podle obrázku 6.3 se vychází z pravidel o součtu úhlu v trojúhelníku. Směr se nastavuje stejným způsobem, jako u vertikální stěny, s tím rozdílem, že pro počítání nového úhlu používám odlišný vzorec. Úhel 4. se spočítán z úhlu 3., jehož rovnice je 180° - 90° - úhel 2.

```
Novy_uhel_pohybu = 36.0f – Stary_uhel_pohybu; //aplikace
všechny úhly dělí hodnotou 10 pro snadnější manipulaci
```

Obrázek 6.5: Počítání úhlu pro odraz od horizontální stěny

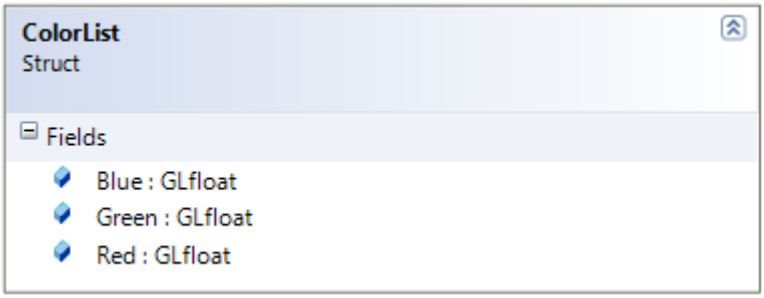
Samotný pohyb objektu se uskutečňuje pomocí změny x-ové a y-ové souřadnice. Změna je počítána za pomocí goniometrických funkcí a vychází z pravidel o trojúhelnících.

```
Novy_x = cos(uhel_pohybu)*rychlost*casova_zmena;
Novy_y = sin(uhel_pohybu)*rychlost*casova_zmena;
```

Obrázek 6.6: Počítání nových souřadnic x a y

6.4 Třída Block

Překážky jsou nedílnou součásti hry. Třída Block obsahuje přetížený konstruktor `Block(GLfloat x, GLfloat y, GLfloat height, int color, int character)`, který je volán při každém vytvoření objektu. První dva parametry určují pozici překážky ve scéně, `GLfloat heigh` určuje výšku (*velikost*) překážky, `int color` určuje typ vykreslovací barvy. S tím, že pro barvu je vytvořena speciální strukturu `ColorList`, která nese informaci o každá barevné složce:



Obrázek 6.7: Struktura pro typ barev

Poslední parametr `int character` udává vlastnost překážky. Nabývá až 5 hodnot:

- 1 – zničitelné jednou ranou
- 2 – vydrží první ránu a změní barvu na zelenou
- 3 – vydrží první ránu a změní barvu na modrou
- 4 – vydrží první ránu a změní barvu na červenou
- 5 – nezničitelná

O vykreslení se stará metoda `void Draw(int Type)`, kde parametr `int Type` určuje, zda se má vykreslit překážka pro lehkou nebo těžkou obtížnost, jak je popsáno v kapitole 5.3.2.

6.5 Třída Projectile

V kapitole 5.3.1 jsou popsány vlastnosti jezdce, ze kterých vyplývá možnost používání střeliva. Třída Projectile určuje základní vlastnosti a chování každého náboje. Opět je zde použitý přetížený konstruktor `Projectile(GLfloat x, GLfloat y)`, kde x a y určuji počateční pozici střely.

Pohyb vystřeleného objektu řídí zděděná virtuální metoda `virtual void Update(Uint32 dt)`, kde se zvyšuje pouze y-ová souřadnice na základně rychlosti a časové změny.

6.6 Třída Effect

Většina her má zajímavé efekty pro zvýšení hratelnosti a celkového dojmu z hraní. Proto jsem vytvořil speciální třídu Effect, která obsahuje všechny efekty, které se vyskytují v aplikaci. Základním prvkem je přetížený konstruktor `Effect(GLfloat x, GLfloat y, int Color, int Type)`. Stejně jako u třídy Block určuje `GLfloat x,y` pozici pro vykreslení efektu, `int Color` typ vykreslovací barvy. Pro barvu zde rovněž slouží struktura `ColorList`. Poslední parametr určuje typ efektu. Vše se vykresluje pomocí metody `void Draw()`:

```
void Draw(){
     switch(Type){
          case 1:
               //vykreslení efektu 1
               break;
          case 2:
               //vykreslení efektu 2
               break;
          ...
     }
}
```

Obrázek 6.8: Metoda Draw() pro vykreslování efektů

Zajímavý efekt je tvořen pomocí náklonu celé scény v závislosti na pozici jezdce. Díky tomu hra působí dynamickým a aktivním dojmem.

6.7 Třída Symbol

Velké množství aplikací potřebuje určitým stylem pracovat s písmeny a fonty. Z důvodu, že aplikace používá SDL, mohl jsem pro texty využít služeb knihovny SDL_ttf. Tato knihovna se bohužel ukázala v kombinaci OpenGL a stylem mé aplikace jako nepraktická a vytvořil jsem zcela nový vlastní font pomocí OpenGL primitiv. Třída má pro symboly (*písmeno české abecedy*, *číslice*, *závorky*) jednotlivé metody, ve které je každý symbol ručně vykreslen. Pro urychlení práce se

všechny symboly vykreslí v konstruktoru do zobrazovacího seznamu, ze kterého jsou jednotlivě zobrazovány celé posloupnosti písmen, číslic a symbolů pomocí příkazu `glCallLists(strlen(Text), GL_BYTE, (GLbyte *) Text).`

6.8 Třída Wall

Třída Wall vykresluje základní herní prostředí (*pozadí, stěny, menu ve hře*). Pozadí hry se skládá ze sítě, která je vykreslena do zobrazovacího seznamu pro rychlejší práci. Každá úroveň používá jiné zbarvení. I zde je použita vlastní struktura `ColorList` pro určování barevného tónu celé sítě.

6.9 Třída Menu

Pro lepší orientaci ve hře slouží úvodní obrazovka neboli herní menu. Veškeré ovládání je řešeno pomocí klávesnice a písmen, které jsou vždy označené v hranaté závorce. Menu má několik hlavních části (*prvků*):

- **[H]RÁT**: Uživatel spustí hlavní hru s příběhem
- **[P]ŘÍBĚH**: Základní popis příběhu a uvedení uživatele do problematiky
- **[O]VLÁDÁNÍ**: Seznámení se základními prvky hry
- **[A]UTOR**: Informace o autorovi a práci
- **[N]AHRÁT**: Hrát uloženou pozici
- **[R]ESET**: Zrušení právě hrané hry
- **[K]ONEC**: Ukončení celé aplikace
- **[ČISLICE]:** Hrát jednotlivé úrovně
- **[L]EHKÁ**: Nastavení obtížnosti na lehkou úroveň
- **[T]ĚŽKÁ**: Nastavení obtížnost na těžkou úroveň

Celé menu má vždy jeden barevný tón. Při každém spuštění aplikace je vygenerováno náhodné číslo pomocí funkce `rand ()` a následovně naplněna struktura `ColorList` podle vygenerovaného čísla, které určuje zbarvení všech prvků v menu. Volí se základní barvy z modelu RGB a CMY.

6.10 Třída Text

Mezi hlavní prvky aplikace patří rozvíjející se příběh, kde je mnoho textů. Vytvořil jsem třídu, která spravuje a zobrazuje veškerý text. Text je vykreslován vždy po řádcích pomocí metody `TextStyle(GLfloat X,GLfloat Y,GLfloat Scale,GLfloat Width,GLfloat Alpha,char* Text)`, kde první dva parametry určují pozici řádků. `GLfloat Scale` udává

velikost textu (*písma*), `GLfloat Width` nastavuje šířku písma, `GLfloat Alpha` intenzitu průhlednosti a `char* Text` je samotný text, který chci vypsat ze zobrazovacího seznamu. Také zde používám strukturu `ColorList`, jelikož v jednotlivých úrovních se mění barva textu.

6.11 Třída Sound

Kompletní zvukový systém aplikace je tvořen za pomocí knihovny SDL_mixer. Knihovna se stará o načítání zvuků (`Mix_LoadWAV(const char *soubor)`), které jsou ve formátu wav a načítání hudby (`Mix_LoadMUS(const char *soubor)`) ve formátu mp3. Každý zvuk je nahrán do jednotlivé proměnné typu `Mix_Chunk*`. Pro soubory s hudbou jsem vytvořil speciální pole `Mix_Music *Music[28]`, do kterého nahraji jednotlivé skladby. Všechny zvuky a hudba se při prvotní inicializaci nahraje do paměti. Jednotlivé úrovně mají vlastní melodii, která má dvě části. Začátek a smyčku pro přehrávání stále dokola. Z důvodu aby se zvuky nepřekrývaly a přehrály se vždy celé, tak každý zvuk má svůj vlastní zvukový kanál, kde se přehrává.

6.12 Třída GameEngine

Třída GameEngine tvoří celý herní systém, zapouzdřuje všechny používané objekty a pracuje s nimi. Pomocí téhle třídy se inicializují a vykreslují úrovně, počítá kolizní systém, ukládá a načítá hra, pracuje s SDL událostmi. Hlavními metodami jsou `MainLoop()` a `EngineEvent()`. První jmenované metoda má na starosti hlavní jádro celé aplikace, nachazí se zde nekonečná smyčka `while(1)`, vypočítává se časová změna za pomocí funkce `SDL_GetTicks()` a vykreslují úrovně. Druhá jmenovaná metoda pracuje s SDL událostmi a klávesnicí.

6.12.1 Inicializace úrovní a práce s efekty

Inicializace jednotlivých úrovní probíhá pomocí metod `EngineInitLevel_X()`, kde X znamená číslo úrovně. Pro zapamatování pozice, vlastností a barvy každé překážky v úrovni (*třída Block*) jsem vytvořil čtyři C++ kontejnery `std::list<Block> B1,B2,B3,B4`, z důvodu rozdělené herní scény na čtyři části, abych při počítání kolízí procházel pouze určitý kontejnér v závislosti na pozici míčku nebo střely. Pomocí C++ funkce `push_back()` nahraji všechny překážky do připravených kontejnérů za pomocí přetíženého konstruktoru třídy Block, který určuje vlastnosti jednotlivé překážky jako je pozice, barva a charakter.

Ve hře se vyskytuje mnoho efektů, proto bylo důležité vytvořit způsob, jakým systémem budou vykresleny. Systém je podobný jako u inicializace úrovní. Vytvořil jsem si C++ kontejner

`std::list<Effect> E`, kde také pomocí funkce `push_back()` a přetíženého konstruktoru třídy Effect nahraji efekt, který vznikl. Stejným způsobem pracuji i se střelami, které se nahrávají do kontejneru `std::list<Projectile> P`.

6.12.2 Vykreslování úrovní

Každá úroveň má svou metodu pro vykreslení `EngineDrawLevel_X()`, X opět znamená číslo úrovně. Zde se volají všechny vykreslovací metody `Draw()`, které se vyskytují u jednotlivých objektů. O vykreslení elemetů uložené v C++ kontejnérech (*překážky*, *efekty*, *střely*) se stará metoda `EngineThrough()`.

```
for(std::list<Effect>::iterator it = E.begin(); it != E.end(); it++){
	it->Draw();
}
for(std::list<Projectile>::iterator it = P.begin(); it != P.end(); it++){
	it->Draw();
}
for(std::list<Block>::iterator it = B1.begin(); it != B1.end(); it++){
	if(this->Difficulty == 1){
		it->Draw(1);
	}else if(this->Difficulty == 2){
		it->Draw(2);
	}
}
```

Obrázek 6.9: Ukázka metody EngineThrough(), která vykresluje elementy z C++ kontejnerů

7 Testování a uživatelské hodnocení

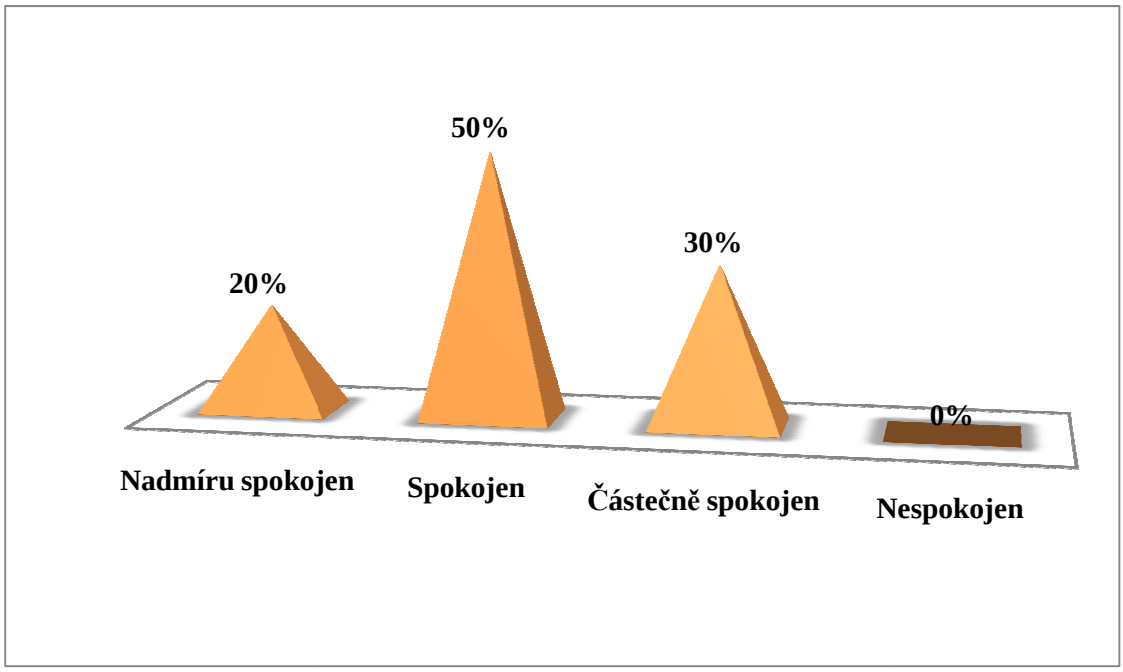
Nedílnou součásti vývoje aplikace, speciálně vývoje počítačové hry, je testování. Použil jsem dvě základní metody, black-box (*metoda černé skřínky*) a white-box (*metoda bíle skřínky*)[14]. V prvním stadiu vývoje bylo hlavně zapotřebí najít základní chyby, které jsem odhalil pravidelným hraním a aplikoval jsem metodu black-box, jelikož mě zajímaly vlastnosti a funkčnost aplikace. Poté jsem mohl aplikaci nabídnout uživatelům pro testování a sbírání jejich reakcí. Po týdenním zkoumání, vyšlo najevo, že základním problémem je počítání skóre a kolizního systému. Jakmile jsem posbíral potřebné údaje, mohl jsem použit metodu white-box a zkoumat chyby v kódu a v samotných algoritmech. Na základě poznatků od uživatelů a výsledku testování metodou white-box jsem návrh plně přepracoval a navrhl nový kolizní systém, který je popsán v kapitole 5.4. Původní systém pouze porovnával x-ové a y-ové souřadnice všech elementů ve hře a počítání skóre je závisle na kolizním systému.

Když byla aplikace opravena, opět prošla testováním samotných uživatelů. Výsledek byl už pozitivní, ale všechny reakce mířily na jedno místo, a to obtížnost. Původně aplikace obsahovala pouze jednu obtížnost a to současně těžkou. Mnoho uživatelů, kteří nemají zkušenosti s hraním her, si stěžovali na problémy dokončit alespoň dvě úrovně. Jelikož hra je určena hlavně pro zábavu uživatelů, rozhodl jsem se změnit návrh a implementovat lehkou obtížnost, kde překážky mají větší velikost (*viz. 5.3.2*). Změna se setkala s pozitivními ohlasy a byly dopracovány zbývající prvky aplikace.

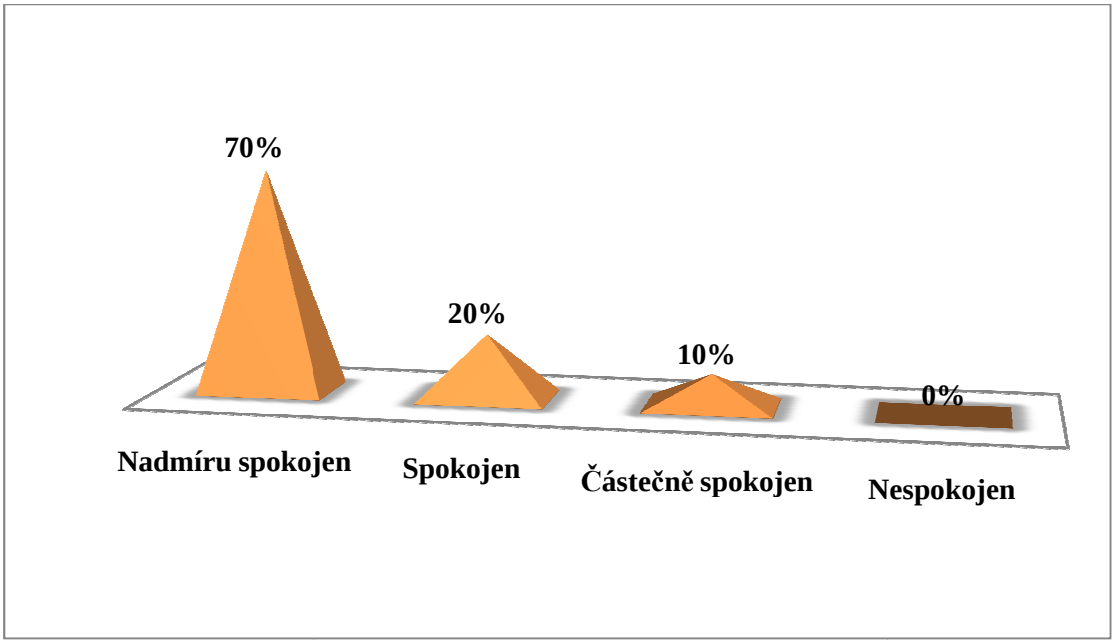
Jelikož je aplikace implementována pomocí knihoven, které jsou podporovány ve více systémech (*Windows, Linux*), prováděl jsem testování i na ostatních platformách. Výsledkem je, že aplikace pracuje jak pod systémem Windows, tak i pod systémem Linux.

7.1 Celkové hodnocení uživatelů

Aplikace byla předložena deseti uživatelům ve věku od 15 – 60 let, aby následně vyplnili dotazník s hodnocením celé aplikace. První testování mělo starý kolizní systém a špatné počítání skóre. Po následné opravě a vyladění chyb se hodnocení aplikace zvýšilo. Uživatelé hodnotili celkový dojem ze hry, vytvořenou hudbu, zvuky a styl příběhu.



Obrázek 7.1: Hodnocení uživatelů po první sérii testování se starým kolizním systémem



Obrázek 7.2: Hodnocení uživatelů konečné verze s novým kolizním systémem a dvěma obtížnostmi

8 Závěr

Cílem práce bylo vytvořit smysluplnou a zábavnou počítačovou hru s historickými grafickými prvky.

Práce analyzuje historii počítačových her (*kapitola 2*), vývoj a rozdělení počítačové grafiky a popis grafických knihoven DirectX a OpenGL. Následující kapitoly (*5*,*6*) diskutují o návrhu a programovém řešení aplikace. Závěr práce se zaobírá testováním hry a sbíráním údajů od uživatelů.

Aplikace vychází ze známé hry Arkanoid a je implementována s použitím objektového návrhu jazyka C++ a knihoven OpenGL a SDL. Ve hře se oproti originálu vyskytuje rozšíření ve formě příběhové linie. Díky tomu se podařilo vytvořit relativně plnohodnotnou počítačovou hru.

Hra byla úspěšně testována na systémech Windows (*přesněji Windows XP a Windows 7*) a Linux (*distribuce Ubuntu*). Během vývoje aplikace byla zpětná vazba (*hodnocení*) získávána od deseti uživatelů. Díky tomu byly ve hře prováděny určité úpravy, popsány v kapitole 7.

Práce mi detailně přiblížila všechny klady a zápory při tvorbě návrhu a vývoji počítačové hry, jelikož tento obor byl donedávna mou velkou neznámou.

8.1 Možné rozšíření

Možnosti pro rozšíření se nabízí celá řada. Kromě vytváření nových úrovní a herních módů bych se rád věnoval implementaci shaderů. Vertex a fragment shadery bych používal pro grafické efekty (*rozmazání obrazu, možnost nasvícení scény*). Geometry shadery bych rád později použil pro nasimulování fyzikálních zákonů jako je gravitace a na ní závislý pohyb všech elementů ve hře. V pozdější fázi je mým cílem implementovat případně síťové rozhraní.

Použitá literatura

[1] Shreiner, D.; Woo,W.; Meider, J.; Davis,T.: *OpenGL: Průvodce programátora*. 1. vyd. Brno: Computer Press, a.s., 2006. 679 s. ISBN 80-251-1275-6

[2] Morrison, M.: *Naučte se programovat počítačové hry za 24 hodin*. 1. vyd. Brno: Computer Press, a.s., 2004. 421 s. ISBN 80-251-0371-4

[3] Historie počítačových her I. [Online, navštíveno 9.4.2010] URL http://www.high-voltage.cz/2008/historie-pocitacovych-her-i

[4] Sláma, D. Chléb a hry: Historie počítačových her. [Online, navštíveno 10.4.2010] URL http://www.zive.cz/clanky/chleb-a-hry-historie-pocitacovych-her/sc-3-a-147762/default.aspx

[5] Zídek, K. Vývoj počítačové grafiky. [Online, navštíveno 16.4.2010] URL http://www.fi.muni.cz/usr/jkucera/pv109/2006/xzidek2.htm

[6] Carlson, W. A critical history of computer graphics and animation: The emergence of computer graphics technology. [Online, navštíveno 17.4.2010] URL http://design.osu.edu/carlson/history/lesson2.html

[7] OpenGL vedle DirextX stále žije. [Online, navštíveno 25.4.2010] URL http://www.ddworld.cz/linux/opengl-vedle-directx-stale-zije-vychazi-opengl-3-5.html

[8] Čech, D. OpenGL: referát na praktikum z informatiky. [Online, navštíveno 26.4.2010] URL http://nehe.ceske-hry.cz/cl_gl_referat.pdf

[9] Kršek, P. Základy počítačové grafiky IZG [Online, navštíveno 27.4.2010]

[10] Tišnovský, P. Grafická knihovna OpenGL [Online, navštíveno 28.4.2010] URL http://www.root.cz/serialy/graficka-knihovna-opengl

[11] Turek, M. Seriál SDL: Hry nejen pro Linux [Online, navštíveno 29.4.2010] URL http://www.root.cz/serialy/sdl-hry-nejen-pro-linux/

[12] Herout, A. Počítačová grafika PGR [Online, navštíveno 27.4.2010]

[13] Minařík, P. Detekce kolizí v DirectX. [Online, navštíveno 30.4.2010] URL http://programovani.net-mag.cz/?action=art&num=459

[14] Král, L; Hazdra, T. Testování a diagnostika softwaru [Online, navštíveno 4.5.2010] URL http://www.odbornecasopisy.cz/index.php?id_document=27838

Seznam použitých zkratek a symbolů

GNU LGPL (*GNU Lesser General Public License*, typ licence) – licence svobodného softwaru.

GPGPU (*General Purpose computing of Graphics Processing Unit*, obecné výpočty prováděné. grafickým čipem) – Jedná se o rychle rozvíjející se oblast grafických karet. Pro tyto operace existují knihovny jako je CUDA nebo AMD Brook+.

GPU (*Graphics Processing Unit*, grafická procesor) – procesor na grafické kartě zajišťující vykreslování dat na zobrazovací zařízení.

HAL (*Hardware Activation Layer*, vrstva v Direct3D) – volá funkce na konkrétní hardware.

HEL (*Hardware Emulation Layer*, vrstva v Direct3D) – vypočítává DirectX funkce softwarově.

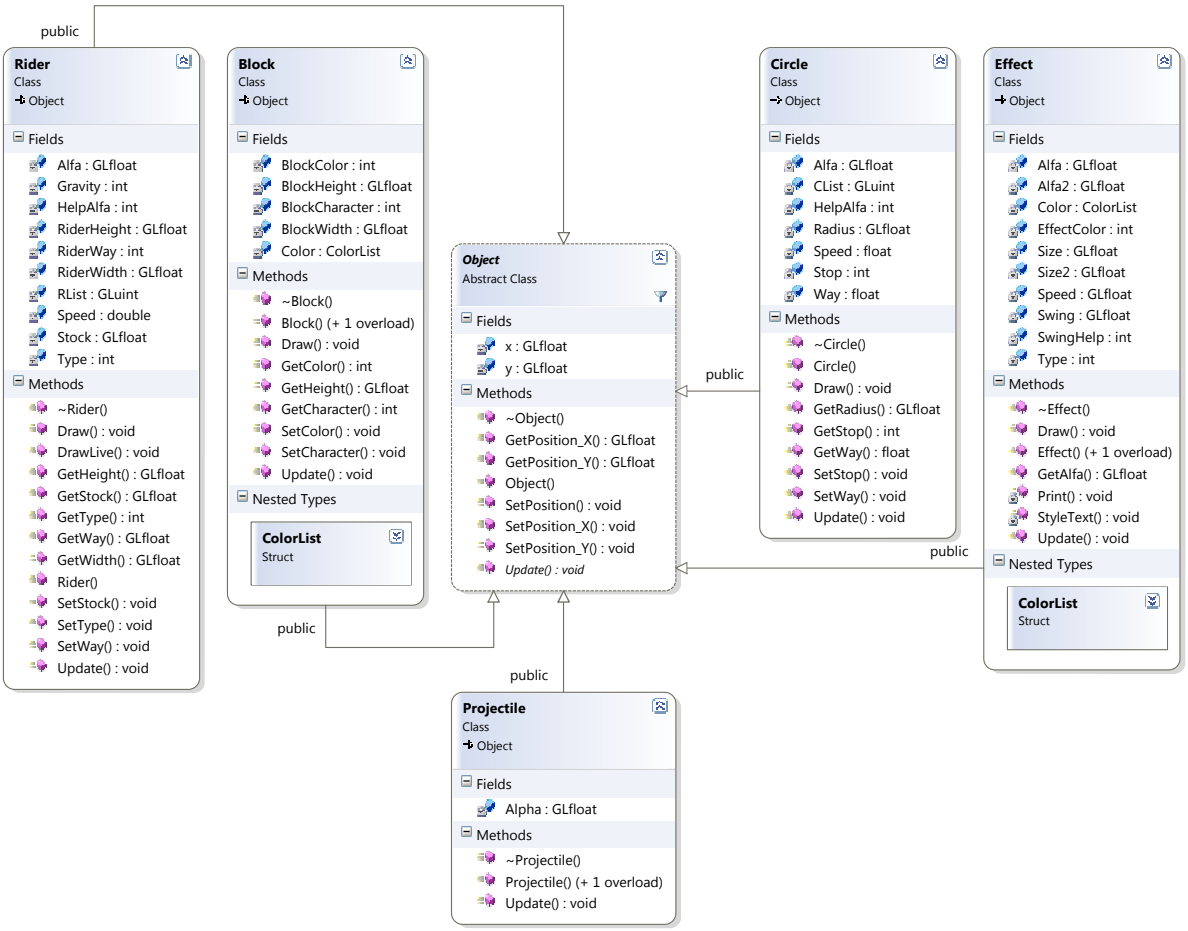
OpenGL (*Open Graphics Library*, grafická knihovna) – volně přístupná grafická knihovna.

SDK (*Software development kit*, sada nástrojů pro vývojáře) – umožňuje lepší práci se softwarovým vybavením. Například jednoduché aplikační programové rozhraní pro programovací jazyk.

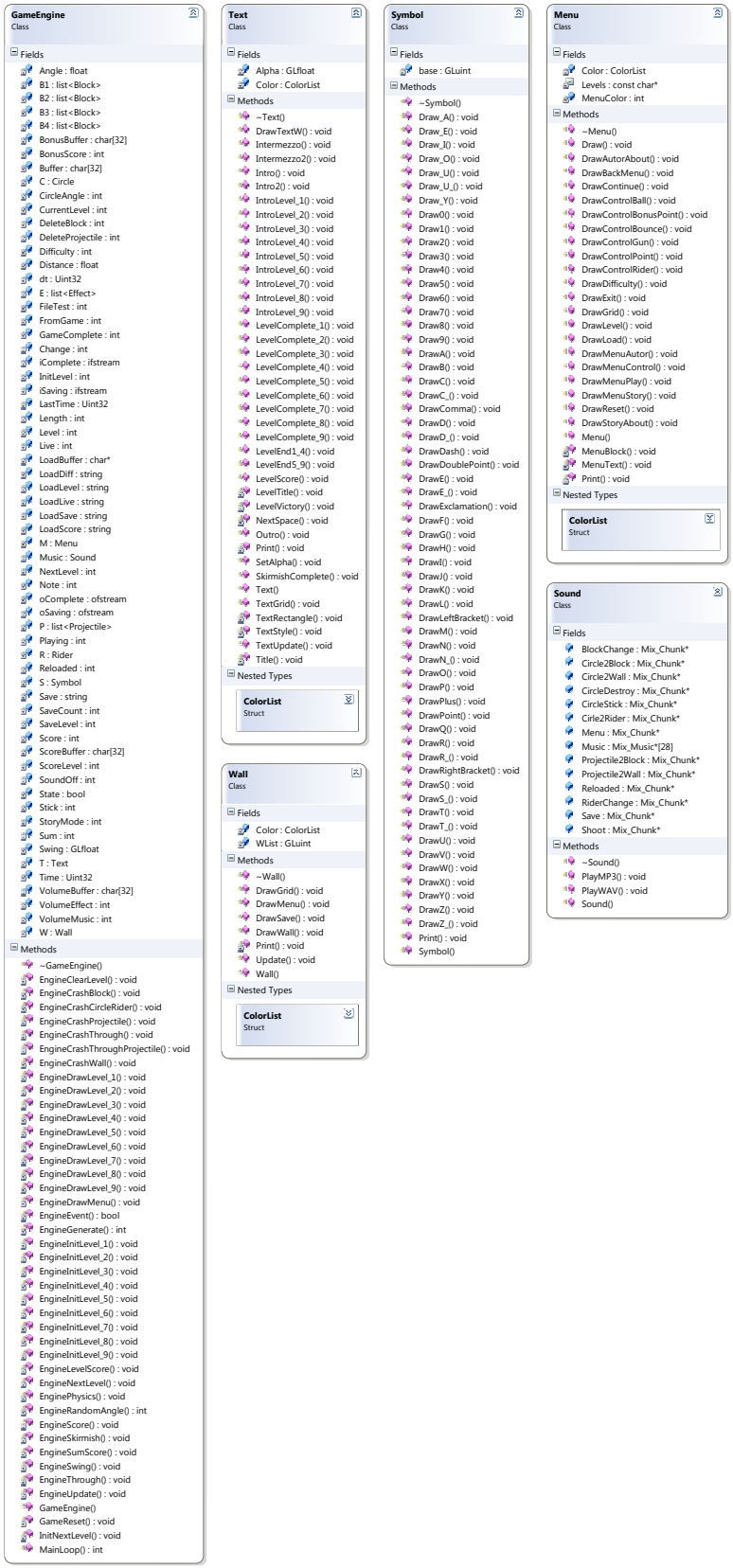
SDL (*Simple DirectMedia Layer*, multimediální knihovna) – multiplatformní multimediální knihovna pro přístup k periferiím a základnímu hardwaru.

Seznam příloh

Příloha 1. Diagram tříd



Obrázek 1.1: První část diagramu tříd



Obrázek 1.2: Druhá část diagramu tříd

Příloha 2. Ovládání a vzhled aplikace

Hlavní nabídka

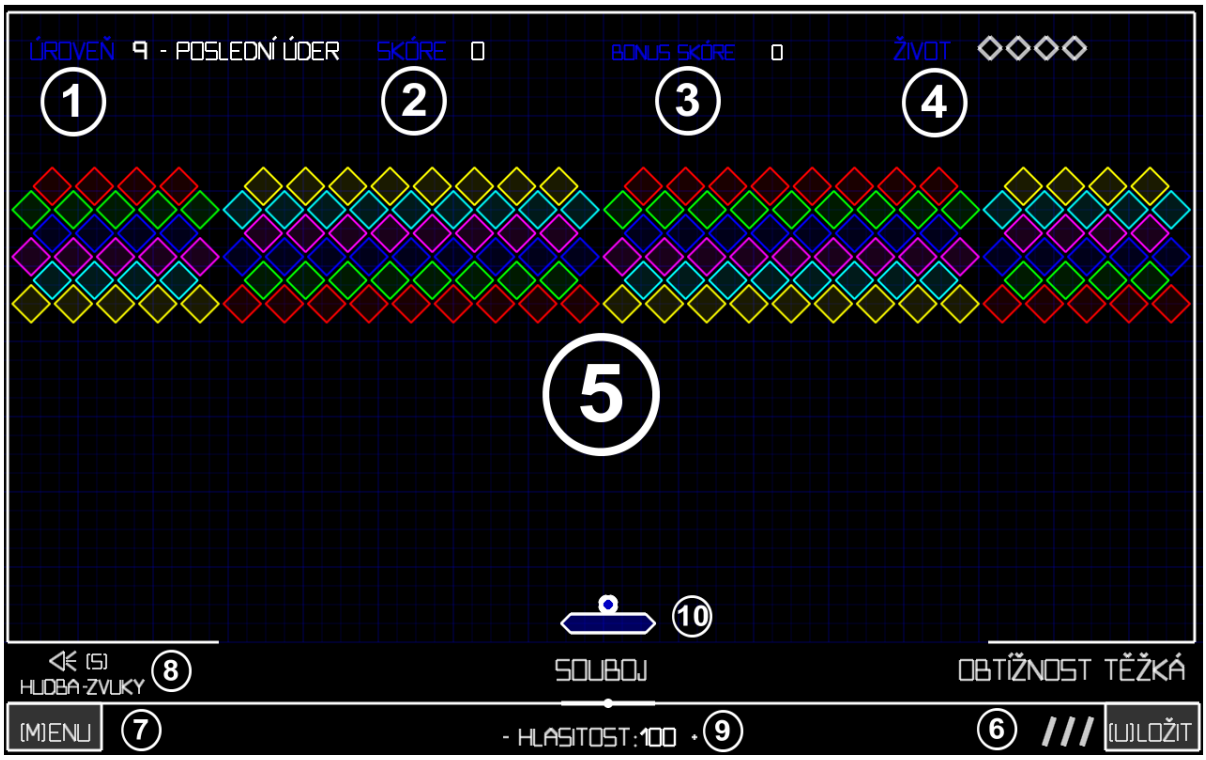
- ***h/H*** – začít hrát příběhovou linii
- ***p/P*** – základní popis příběhu
- ***o/O*** – základní popis ovládání
- ***a/A*** – informace o autorovi
- ***k/K*** – ukončení aplikace
- ***l/L*** – nastavení lehké obtížnosti
- ***t/T*** – nastavení těžké obtížnosti
- ***n/N*** – nahrát uloženou pozici v příběhové linii
- ***r/R*** – zrušit právě probíhající hru
- ***z/Z*** – vrátit se zpět do hry
- ***číslice 1 až 9*** – spuštění zvolené úrovně
- ***s/S*** – zapnutí a vypínaní zvuků a hudby

Intro

- ***mezerník*** – pokračovat dále v příběhu

Hra

- ***šipka doleva*** – pohyb jezdce doleva
- ***šipka doprava*** – pohyb jezdce doprava
- ***mezerník*** – vystřelit míček
- ***šipka nahoru*** – vystřelit náboj
- ***s/S*** – zapnutí a vypnutí zvuků
- **+** - zvyšování hlasitosti
- **-** - snižovaní hlasitosti
- ***u/U*** – uložit požadovanou úroveň (*lze pouze v příběhové linii*)
- ***m/M*** – návrat do hlavní nabídky



Obrázek 2.1: Vzhled herního prostředí

- ***1*** – číslo a název úrovně
- ***2*** – výše Vašeho skóre
- ***3*** – výše Vašeho bonusového skóre
- ***4*** – počet životů
- ***5*** – herní plocha úrovně
- ***6*** – počet možných uložení
- ***7*** – návrat do hlavní nabídky
- ***8*** – zapínání a vypínání zvuků
- ***9*** – ovládání hlasitosti
- ***10*** – jezdec, míček a typ herního módu