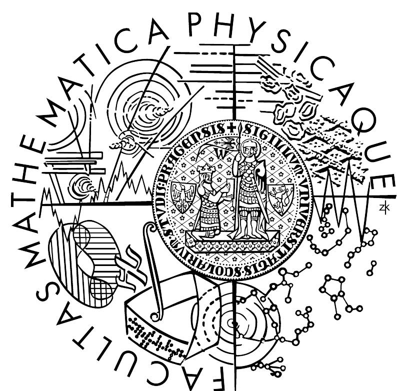
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Andrej Kruták

AnoRaSi - fyzikálně-realistický simulátor v 3D

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Tomáš Tichý
Studijní program: Informatika

2006

Poďakovanie

Na tomto mieste by som chcel poďakovať svojmu vedúcemu bakalárskej práce – Tomášovi Tichému za trpezlivosť, ktorú musel mať vzhľadom na dlhý vývoj aplikácie. Takisto mu ďakujem za pripomienky k samotnej práci.

Ďakujem svojim priateľom za pomoc pri riešení niektorých fyzikálnych a matematických problémov. Im a Pedritte som vďačný za psychickú podporu v posledných mesiacoch písania programu a tejto práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 10.8.2006. Andrej Kruták

Obsah

KAPITOLA 1 – ÚVOD....5

1.1 Cieľ práce....5
1.2 Motivácia....5
1.3 Postup tvorby programu....6
1.4 Hardvérové nároky a ovládanie aplikácie....7
1.5 Inštalácia a kompilácia....7

KAPITOLA 2 – NÁVRH APLIKÁCIE....9

2.1 Zamyslenie nad architektúrou....9
2.2 Organizácia dát....11

KAPITOLA 3 – SERVER / FYZIKÁLNY MODEL....16

3.1 Architektúra servera....16
3.2 Úvod do knižnice ODE....20
3.3 Popis objektov a ich vlastností v ODE....20
3.4 Vytvorenie realistickej simulácie....23
3.5 Jeden krok simulácie....27
3.6 Fyzika automobilu....29
3.7 Hra....33

KAPITOLA 4 – KLIENT / ZOBRAZENIE SIMULÁCIE....35

4.1 Knižnice Irrlicht a OpenAL++....35
4.2 Grafické menu aplikácie....36
4.3 Zobrazovanie simulácie....40
4.4 Aproximácia polôh telies v scéne....44
4.5 Herné prvky scény....44

KAPITOLA 5 – SIEŤOVÁ KOMUNIKÁCIA....46

KAPITOLA 6 – ZÁVER....48

LITERATÚRA....50

PRÍLOHY....51

Název práce: AnoRaSi - fyzikálně-realistický simulátor v 3D
Autor: Andrej Kruták
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Tomáš Tichý
e-mail vedoucího: tichy@math.cas.cz
Abstrakt: Anorasi je program simulujúci fyzické prostredie tvorené automobilmi a okolitým terénom. Táto práca zoznamuje čitateľa s postupom vytvorenia tohoto simulátora. Vysvetľuje nutnosť použitia modelu klient/server ako architektúru aplikácie a následne aj spôsob komunikácie oboch častí tohto modelu medzi sebou cez sieťové rozhrania. Serverová časť aplikácie slúži na vlastné simulovanie fyzického prostredia. Práca popisuje vytvorenie jednotlivých objektov simulácie, spôsob, akým je dosiahnutá ich vhodná vzájomná interakcia, v stručnosti popisuje aj simulované fyzikálne vlastnosti vozidiel a taktiež spôsob akým sa z jednotlivých simulácií vytvárajú závody. Klientská časť slúži na poskytnutie grafickej a zvukovej interakcie medzi užívateľom a serverom. Časť práce sa preto zaoberá aj zobrazovaním prostredia a ozvučením vlastnej simulácie.
Klíčová slova: simulácia, automobil, hra, fyzika

Title: AnoRaSi - physically-realistic simulator in 3D
Author: Andrej Kruták
Department: Department of Software Engineering
Supervisor: RNDr. Tomáš Tichý
Supervisor's e-mail address: tichy@math.cas.cz
Abstract: AnoRaSi is a program simulating physical environment built of vehicles and the surrounding terrain. This work introduces the reader to techniques used to create this simulator. It explains the need of use of client/server model as the application architecture. Subsequently it describes the way both parts of the model communicate through network interfaces. The destiny of server part of the application is to simulate the world in a physical way. Work describes, how the objects of simulation are created and the way their interaction is achieved. In short it also describes simulated physics attributes of vehicles and also, the way how a race is created from each simulation. Client part enables user to interact with server by presenting him the scene visually and acoustically. Part of this work is showing, how to display the scene and create appropriate sound output.
Keywords: simulation, vehicle, game, physics

Kapitola 1 – Úvod

1.1 Cieľ práce

Cieľom tejto práce bolo vytvoriť program (resp. hru) simulujúci pohyb motorových vozidiel v členitom trojrozmernom priestore. Terén mal byť tvorený objektami tvaru hranola a gule, ktoré by mohli byť ľubovoľne umiestnené a otočené. Prostredie ale aj samotné vozidlá by mali mať do veľkej miery konfigurovateľné fyzikálne vlastnosti, najmä čo sa týka hmotností a povrchov ich súčastí.

Ďalším cieľom bolo umožnenie vzájomného prepojenia viacerých účastníkov – vytvorenie automobilových pretekov, v ktorom každý z účastníkov ovláda svoje vozidlo.

Dôležitý pre používateľov je aj vzhľad a jednoduchosť použitia aplikácie. Jej ovládanie teda malo byť intuitívne už od úvodných obrazoviek, ktoré by umožňovali konfiguráciu najdôležitejších vlastností aplikácie. Aj ovládanie vozidla počas simulácie malo okrem byť, okrem istej miery realistickosti, pre užívateľov pohodlné.

Požiadavkom bolo aj použitie takých vývojových prostriedkov – knižníc, ktoré by neznemožňovali spustenie softvéru na rôznych druhoch operačných systémov. Minimálne mal byť program preložiteľný a spustiteľný na systémoch Microsoft Windows a na systémoch s jadrom Linux.

1.2 Motivácia

Inšpiráciou k zvoleniu tejto témy práce boli už existujúce počítačové hry simulujúce závody automobilov. Nie je triviálne simulovať pohyb áut v 3D priestore – a kvalitných návodov, resp. informácií o tom, ako takúto simuláciu vytvoriť, je dokonca aj na internete nedostatok. Hlavným dôvodom vypracovania teda bolo oboznámenie sa s postupmi využívanými na vytvorenie realtime simulátorov a ich následné zverejnenie formou tohto dokumentu a demonštrovanie programom.

Existuje viacero funkčných implementácií simulátorov rôznej kvality. Ako príklad môžeme uviesť napríklad veľmi kvalitné komerčné simulátory rady F1 od firmy Electronic Arts, ktoré sú, ako z názvu vyplýva, orientované na simuláciu závodov F1.

Príkladom môže byť taktiež ďalší veľmi kvalitný komerčný projekt spoločnosti MGI: Viper Racing. Na internete je možné nájsť mnoho, aj keď často nekvalitných, opensource projektov. Najlepšími a najrealistickejšími z nich sú Racer (www.racer.nl), TORCS (torcs.sf.net), Vamos (vamos.sf.net) a Vdrift (vdrift.net).

Ani jeden z nich však nepoužíva rovnakú kombináciu prostriedkov (knižníc) ako mala používať finálna verzia AnoRaSi. Týmito knižnicami sú Irrlicht (určená na vykresľovanie scény), OpenAL (zvukový výstup) a ODE (univerzálna knižnica starajúca sa o simuláciu fyzikálneho sveta).

Okrem toho každý zo zmieňovaných projektov má určité negatívne vlastnosti, ktoré ovplyvňujú jeho použiteľnosť. Posledné dva projekty napríklad nesimulujú podstatnú vlastnosť motora – jeho brzdiaci účinok v niektorých situáciách. Zvyšné projekty sú už vcelku realistické, no kvôli uzavretosti zdrojových kódov respektívne kvôli prílišnej zložitosti je ťažké zistiť princíp ich presného fungovania. Ďalší z cieľov práce teda bol vyhnúť sa týmto chybám a čo najprehľadnejšie implementovať program – aby z jeho kódu bolo možné jednoducho „vyčítať“ aspoň najdôležitejšie algoritmy.

1.3 Postup tvorby programu

Po schválení špecifikácie programu bola navrhnutá architektúra a začali sa práce na samotnom kóde. Autor mal na začiatku projektu takmer nulové skúsenosti s už uvedenými knižnicami. To, a malé množstvo informácií o fungovaní realistických simulátorov spôsobilo, že prvé mesiace tvorby programu boli strávené učením sa ako dané knižnice používajú a akým spôsobom je možné ich prepojenie do jedného celku. Ďalším problém boli stratené vedomosti o vektorovej algebre a použitie kvaterniónov knižnicou ODE na určovanie rotácie telies v simulácii. Najdôležitejšie informácie o tejto problematike boli nájdené vo vynikajúcej publikácii o počítačovej grafike [1].

Dôležitým krokom bolo značne rozsiahle prepísanie aplikácie po pridaní sieťového kódu. Od tohto okamihu bolo už potrebné počítať s časovým oneskorením a prípadnými výpadkami spojenia medzi klientmi a serverom. To sa prejavilo napríklad potrebou implementovať algoritmy aproximácie pozícií objektov či použitie nepriamych odpovedí servera[1].

[1]Tým sa myslí najmä sieťová komunikácia v štýle protokolu X windows: Na správy sa neodpovedá, kým to nie je nevyhnutné. Okrem toho bolo kvôli plynulosti behu programu potrebné zabrániť aktívnemu čakaniu, keď by sa očakávala odpoveď z druhej strany sieťového spojenia – a zdržovalo tak vykonávanie iného kódu.

Netriviálnou úlohou bolo aj vytvorenie (pomocou veľmi obecnej knižnice ODE) realisticky sa chovajúceho prostredia – spolu so simuláciou všetkých dôležitých aspektov vozidiel – motora, prevodového ústrojenstva, aerodynamiky atď.

1.4 Hardvérové nároky a ovládanie aplikácie

AnoRaSi funguje na princípe klient-server aplikácie. To v praxi prináša možnosť spustiť server (simulujúci fyzikálne prostredie, využívajúc najmä výkon procesora) na výkonnejšom počítači, kým klienti (zobrazujúci simuláciu užívateľovi, k čomu potrebuje len rýchlu grafickú kartu) môžu používať aj menej výkonné a zásadne to neovplyvní presnosť simulácie. Vlastnosťou aplikácie dôležitou pre určenie požiadavkov na hardvér počítača je to, že na jednom z počítačov musí bežať zároveň server aj klient.

Odporúčaná konfigurácia je teda určená pre súčasné spustenie oboch súčastí na jednom počítači a v prípade samostatného spustenia klienta sa tieto nároky môžu mierne znížiť (hlavne čo sa týka nárokov na procesor). V tabuľke 1. je uvedená orientačná zostava PC, na ktorom by malo už byť možné aplikáciu rozumne používať.

Tabuľka 1: Konfigurácia PC vhodná na spustenie programu

Komponent PC	**Optimála konfigurácia**
Procesor	Pentium II/600 Mhz, vhodnejší je však aspoň Pentium III/1 GHz
RAM	256 MB, radšej však 512 MB
HDD	~ 30 MB
Grafická karta	Výkonná 3D AGP/PCIe karta, s vlastnou pamäťou minimálne 16 MB
Zvuková karta	Ľubovoľná novšia PCI/integrovaná na doske
Sieťová karta	10/100 Mbit sieťová karta, resp. ISDN/DSL modem s nízkou latenciou

Aplikácia je, čo sa užívateľského prostredia týka, rozdelená na dve časti – menu a samotné zobrazovanie simulácie. Popis ovládania oboch častí spolu s postupom pri spájaní viacerých hráčov je uvedený v užívateľskej dokumentácii aplikácie (ktorá je prílohou tejto práce).

1.5 Inštalácia a kompilácia

AnoRaSi je naprogramovaná tak, aby mohla byť spustená na viacerých platformách. Jedinými platformami, na ktorých však bola testovaná, sú Windows a Linux. Inštalácia

programu na prvej z platforiem spočíva v jednoduchom rozbalení distribučného archívu (ktorý je umiestnený na priloženom médiu pod názvom anorasi.zip). V systéme linux je situácia menej užívateľsky prívetivá a je potrebné aplikáciu ručne skompilovať (kompilácia programu je samozrejme možná aj v systéme windows).

Pred samotnou kompiláciou projektu je potrebné mať pripravené – nainštalované – všetky používané knižnice. V tabuľke 2. sú uvedené použité knižnice spolu s webovými stránkami, kde ich je možné stiahnuť.

Tabuľka 2: Použité knižnice

Irrlicht	http://irrlicht.sf.net
OpenAL++	http://alpp.sf.net
ODE	http://ode.org
OpenThreads	http://openthreads.sf.net

V oboch systémoch je potrebné tieto knižnice skompilovať a nastaviť kompilátory tak, aby boli pripravené ich použiť. V systéme Ubuntu linux je situácia mierne jednoduchšia a knižnice OpenAL++, ODE a OpenThreads sú dostupné ako developer balíky priamo v repozitároch distribúcie. Stále je však potrebné skompilovať knižnicu Irrlicht „ručne“.

Potom ako sú všetky knižnice nainštalované (a v kompilátoroch nastavené cesty k nim), môže dôjsť ku kompilácii samotnej aplikácie. Pod Windows stačí v prostredí Visual Studio otvoriť projektový súbor a spustiť kompiláciu. Pre systém linux je v koreňovom adresári balíku so zdrojovými kódmi pripravený jednoduchý makefile súbor, ktorý na kompiláciu projektu používa prekladač gcc. Po zavolaní programu make sa vďaka tomuto súboru (za predpokladu prítomnosti vyžadovaných knižníc) automaticky skompilujú potrebné súbory a vygeneruje výsledná binárka.

Kapitola 2 – Návrh aplikácie

Celá nasledujúca kapitola je venovaná myšlienkovým postupom, ktorými bola zvolená architektúra výslednej aplikácie. Okrem toho popisuje základné princípy tvorby a umiestnenia jej dátových a konfiguračných súborov.

2.1 Zamyslenie nad architektúrou

AnoRaSi bola od začiatku vyvíjaná tak, aby umožňovala prepojenie viacerých užívateľov. Do úvahy dva modely prepojenia:

Prvým prístupom je štandardný klient-server model. Server v tomto prípade prijíma pripojenia a dáta od klientov, umožňuje klientom navzájom komunikovať (pričom server má úlohu prostredníka), simuluje fyzikálne prostredie a rozosiela jednotlivým klientom naspäť informácie o stave simulácie. Jednotliví klienti sa pripájajú k jedinému serveru a počas simulácie mu zasielajú mu informácie o ovládaní svojho vlastného vozidla. Ďalšou ich úlohou je prezentovať dáta prijaté zo servera užívateľovi aplikácie.

Druhou možnosťou je model distribuovaného výpočtu s jedným hlavným serverom, kde by sa každý z účastníkov „tváril" zároveň ako klient aj ako server. Simulácia by teda prebiehala na každom zo zúčastnených počítačov súčasne. Veľmi pravdepodobne by ale časom dochádzalo k odlišnostiam v „predstave simulácie" u jednotlivých klientov (vzhľadom na odlišnosť použitých hardvérových komponentov a rôznych náhodných javov). Preto by jeden z nich musel fungovať ako „hlavný" server, podľa ktorého sa v určitých intervaloch synchronizujú ostatné. Tento postup by zrejme vyžadoval veľmi veľa algoritmov použitých na sofistikovanú synchronizáciu, pritom by však neviedol k znateľným výhodám. Naopak, každá z klientských staníc by musela mať zároveň výkon na vykresľovanie aj na simuláciu prostredia.

Vzhľadom na tieto a ďalšie skutočnosti bol pre aplikáciu vybraný prvý model. Okrem toho sa prihliadalo aj k zrejmému pozorovaniu, že každý z užívateľov bude s najväčšou pravdepodobnosťou pripojený z iného počítača - všetci teda budú zrejme musieť komunikovať cez nejaký druh siete. Rozšírenosť a všeobecné rozšírenie protokolu TCP/IP na najväčšej sieti sveta – internete – prispelo k výberu práve tohto protokolu ako základného komunikačného prostriedku medzi jednotlivými klientmi.

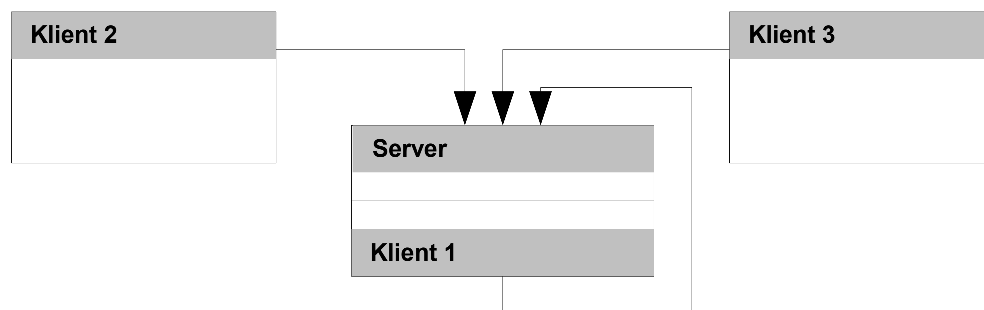
Zvolením modelu klient-server však návrh architektúry nekončí. Vzhľadom na výber protokolu bolo možné, že výsledná aplikácia bude používaná na sieťach

s nedostatočnou priepustnosťou a/alebo priveľkou latenciou. V prípade, že by boli dáta medzi klientom a serverom prijímané a odosielané synchrónne s vykonávaním vlastnej činnosti (na strane klienta je to vykresľovanie prostredia, server vykonáva simuláciu), mohlo by dochádzať k prerušovaniu činnosti. Príkladom by mohla byť komunikácia klienta so serverom. V prípade pomalej linky by mohlo trvať aj niekoľko desiatok milisekúnd, kým by bola prijatá jedna celá správa – a počas celého toho času by klient musel čakať na sieť, kým dané dáta dorazia. Kvôli tomu by nemohol vykresliť nové (hoci aproximované) polohy objektov, čo by si užívateľ s určitosťou vnímal ako negatívny jav. Situácia by bola o to horšia, že sa, kvôli plynulosti animácie, predpokladajú desiatky správ prijaté od servera za sekundu. Podobný problém je aj s odosielaním dát.

Kvôli vyhnutiu sa týmto problémom bol do architektúry aplikácie pridaný ďalší stupeň: oddelenie sieťovej komunikácie od vykonávania ostatnej činnosti, a to tak na strane servera ako aj na strane klienta. Toto oddelenie bolo implementované pomocou vlákien. Každý sieťový komunikačný kanál je umiestnený do samostatného vlákna a dáta, ktoré prijíma, sa ukladajú do zásobníkov. Čo sa odosielaných dát týka, sú dáta bufferované na len strane servera – predpoklad totiž je rádovo menšia náročnosť na priepustnosť odchodzích dát od klienta.

Presunutím simulácie do servera sme síce dosiahli viacerých výhod – napríklad odpojenie jedného z klientov neohrozí simuláciu (v zmysle jej úplného zastavenia). Z pohľadu užívateľa to však také výhodné nie je – na spustenie jedného sedenia by bolo potrebné ručne nakonfigurovať server, spustiť ho a následne sa pripojiť. Takéto riešenie samozrejme nepripadá do úvahy. Preto sú klient aj server umiestnení v tom istom spustiteľnom súbore, a klientská časť umožňuje spustenie a konfiguráciu servera. Na jednom z účastníckych počítačov je teda zároveň spustená klientská aj serverová časť aplikácie. Tým síce čiastočne prichádzame o spomínanú výhodu nezávislosti simulácie na klientoch, na druhú stranu to však prináša jednoduchosť nastavenia a tým aj väčšiu užívateľskú prívetivosť.

Pre názornosť je na obrázku 1. zobrazený schematický diagram komunikácie medzi viacerými počítačmi - jeden rámček v ňom predstavuje jednu spustenú aplikáciu, šípky znázorňujú TCP/IP spojenia.



Obrázok 1: Znázonenie prepojenia viacerých klientov a servera

Konkrétnejšie informácie o sieťovej komunikácii sú uvedené v 5. kapitole, jej obsah však čiastočne naväzuje na informácie z kapitol 3. a 4.

Dôležitou vlastnosťou pri návrhu architektúry programu je aj cieľový systém, na ktorom bude daný program spúšťaný. Zásadným problémom by mohol napríklad byť citeľný nedostatok výpočtového výkonu embedded systémov (PocketPC, mobilné telefóny), alebo nízka výkonnosť grafických kariet (staršie modely PC). Problémom by mohla byť aj zlá podpora realtime aplikácií zo strany operačného systému. Preto boli ako za cieľový systém zvolené počítače triedy PC, fungujúce s operačným systémom Windows alebo Linux. Oba tieto OS poskytujú v podstate identické možnosti tvorby aplikácií a využívania multimediálnych vlastností hardvéru.

V záujme čo najväčšej prehľadnosti kódu programu bol za programovací jazyk aplikácie zvolený C++, ktorý umožňuje vytváranie objektov a ich dedičnosť (dôležitým faktorom pri jeho výbere bolo aj to, že tento jazyk autor práce pozná najlepšie). Objektová orientácia jazyka pomáha udržiavať kód aplikácie dostatočne štruktúrovaný a odstraňuje aj redundanciu kódu.

Tento typ aplikácie priam predurčuje zapuzdrenie algoritmov a informácií do objektov. Príkladom môže byť vozidlo, ktoré sa skladá z viacerých komponentov, pričom tie sú všetky odvodené od typu „pohyblivý objekt“. Rozdelením aplikácie do tried (objektov) dosiahneme výrazné zvýšenie názornosti kódu pri zachovaní v podstate rovnakého výkonu – v porovnaní s neobjektovým riešením.

2.2 Organizácia dát

Okrem samotnej spustiteľnej časti sú dôležitou súčasťou aplikácie aj dáta. Klientská a serverová časť používa rôzne druhy dát.

Klientská časť aplikácie je primárne určená na prezentáciu stavu simulácie užívateľovi. Už pri zobrazovaní menu (úvodného užívateľského rozhrania) je potrebné vykresľovať určitý druh grafiky a ovládacie prvky - väčšinou sa teda jedná o rôzne bitmapy a písma. Dôležitejšia časť je však zobrazovanie samotnej simulácie. Tu už je potrebné načítať samotné modely použitých vozidiel, terénu a ovládacie prvky (napr. poradie v závode, rýchlosť vozidla, otáčkomer motora). Okrem toho užívateľ očakáva aj zvukovú odozvu – napr. „vrčanie motora“.

Naproti tomu serverová časť vyžaduje diametrálne odlišný druh dát. Napríklad nevyžaduje informácie o vzhľade simulovaných objektov – o farbe povrchu, vzorke pneumatík či odtieni oblohy. Dôležité sú iné veci – hmotnosti, tvary modelov a fyzické vlastnosti ich povrchov.

Napriek odlišnostiam v druhu vyžadovaných dát na strane klienta a servera sa jednotlivé údaje dopĺňajú a veľmi často dosť podstatne súvisia. Príkladom môžu byť napríklad statické objekty simulácie. Keďže sa ich pozícia nemení, bolo by zbytočné posielať informácie o nich zo servera klientom, ak by si ich oni mohli automaticky načítať z konfiguračného súboru. Takisto zbytočná by bola potreba uvádzať pozíciu objektov osobitne v konfigurácii servera a klienta. Ak ďalej prihliadneme k faktu, že klient aj server sú vždy zároveň umiestnení na jednom systéme (aj keď užívateľ serverovú časť nemusí použiť), je veľmi výhodné vytvoriť spoločnú konfiguráciu pre klienta aj server. Okrem odstránenia redundancie dát tak zároveň odstránime aj možnú nekonzistentnosť, ktorá by nastala v prípade chybného kopírovania niektorých údajov - pri vytváraní dvojíc konfiguračných súborov.

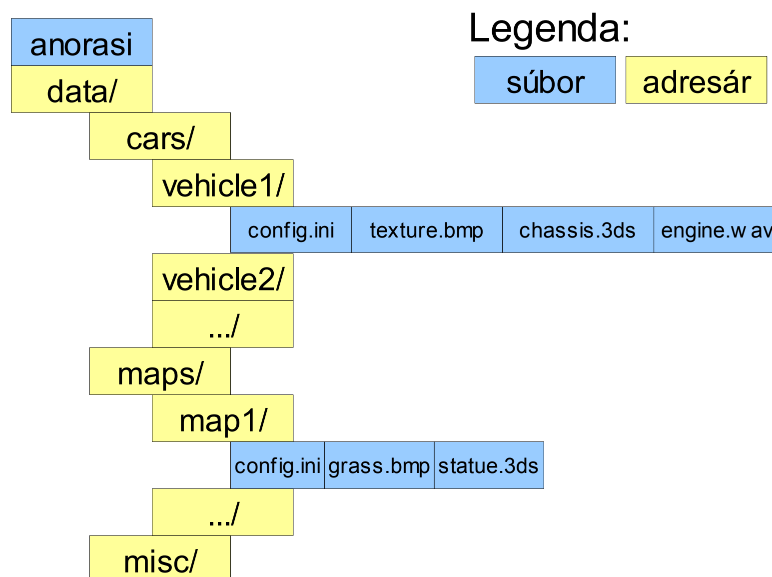
Pre popis každého z objektov v simulácii teda bude použitý najviac jeden konfiguračný súbor. Je však nutné stanoviť, čo budeme považovať za objekty, a ktoré súčasti simulácie zjednotíme do jedného celku. Z povahy simulácie je možné vyčleniť dva hlavné druhy objektov – vozidlá a obklopujúci terén.

Každé vozidlo sa môže skladať z viacerých súčastí. Ak berieme do úvahy súčasti použité v oboch častiach aplikácie (klient a server), sú to najmä karoséria, kolesá a motorová časť vozidla. Je viac ako pravdepodobné, že jednu karosériu ani motor nebude zdieľať viacero druhov vozidiel. Jeden druh kolies sa môže použiť vo viacerých druhoch vozidiel – ale v každom vozidle bude určite inak umiestnené. Okrem toho vlastností, ktoré popisujú koleso, je relatívne málo. Je teda výhodnejšie určiť pre každé vozidlo vlastnosti jeho kolies osobitne (priamo v konfigurácii vozidla) – zavedieme tým

síce určitú redundanciu, v tomto prípade to však príliš nevadí a skôr zjednoduší návrh vozidiel.

Naopak u tratí sa dá predpokladať (keďže sa simulátor snaží o realistickosť), že sa objekty v nich nebudú opakovať a teda každá trať bude tvoriť uzavretý celok. Jej konfigurácia sa teda opäť môže uložiť do jedného súboru.

Napriek tomu, že by sme mohli konfiguráciu oboch druhov hlavných objektov umiestniť do jednotlivých súborov, je zrejmé že okrem konfigurácie budeme o objektoch potrebovať aj ďalšie informácie. U oboch druhov sú to už spomínané textúry, u vozidiel potom napríklad modely karosérií a kolies. Pre prehľadnosť budú oba druhy objektov umiestnené do rôznych adresárov v súborovom systéme, a zároveň pre každý konfiguračný súbor (a teda aj každé vozidlo a každé prostredie/trať) bude vytvorený vlastný podadresár. Okrem toho bude vyčlenený jeden adresár pre dáta všeobecného použitia (kde budú napríklad písma, ale aj textúry použité na vykreslenie hlavného menu a pod.). Výsledná štruktúra adresárov bude teda vyzerať podobne ako na obrázku 2.



Obrázok 2: Štruktúra adresárov aplikácie

Konfigurácia každého z hlavných objektov bude umiestnená v príslušnom adresári v súbore config.ini a všetky dáta, na ktoré sa odvoláva, budú v tom istom adresári.

Okrem už spomínaných konfiguračných súborov AnoRaSi používa ešte ďalšie dva. Prvý je umiestnený priamo v adresári so spustiteľným súborom a volá sa anorasi.ini.

Tento konfiguračný súbor obsahuje nastavenia klienta – napríklad použité grafické rozhranie (OpenGL, DirectX, softvérový mód) a naposledy použitý nick, adresu a port servera, na ktorý sa klient pripája. Druhý súbor sa volá settings.ini a nachádza sa v adresári data/. Nastavenia umiestnené v tomto súbore sú dosť heterogénne – je tu nastavenie niektorých vlastností klientskej (konkrétne ovládacie prvky vozidla – tachometer, otáčkomer a zobrazenie prevodového stupňa) aj serverovej časti (definície vlastností povrchov použitých v jednotlivých prostrediach a vlastností kontaktov medzi týmito povrchmi).

Všetky spomínané konfiguračné súbory sú textové, a majú jednotný hierarchický formát[2]. Pritom táto hierarchia môže mať ľubovoľný počet koreňov a dáta môžu byť uložené v ktorejkoľvek úrovni. Najjednoduchšie sa dá formát popísať na príklade:

```
1   Terrain
2       Object_1
3           type=object #optional
4           file=map1.3ds
5           scale=1 1 1
6           type=multipart #can be also singlepart
7           Physics
8               1
9                   surface=asphalt
10              2
11                  surface=grass
12      Object_2
13          #skip=yes
14          type=water
```

Znaky # označujú začiatok komentárov - za týmto znakom sú všetky znaky až do konca riadku ignorované. Každý riadok má buď tvar „Názov“ (vrchol), alebo „Názov=hodnota“ (list), a je odsadený určitým počtom znakov tabulátor (ASCII kód 9). Prvý tvar označuje vrchol; všetky nasledujúce riadky s väčším počtom tabulátorov na začiatku (ako má ten aktuálny) tvoria jeho potomkov - vrcholy a listy[3]. Definícia podstromu končí na riadku s rovnakým alebo menším počtom tabulátorov (alebo koncom súboru). Príkladmi podstromov sú riadky 1-11, 2-11, 7-11 a 12-14. Cesta k listu sa potom v programe označuje ako postupnosť názvov jednotlivých (pod)koreňov

[2]Dnes už všadeprítomný štandardný hierarchický formát XML nebol použitý z dôvodu prílišnej zložitosti a kvôli potrebe používať kvalitný parser, ktorý by v tomto prípade zbytočne zvyšoval nároky na preklad aj údržbu kódu.

[3]Každý z potomkov musí mať (z technických dôvodov) v rámci danej úrovne podstromu jedinečné meno. Ak je potreba popísať viacero objektov v jednej úrovni, používa sa konvencia Názov_N, kde N sú prirodzené čísla. Pritom kontrola existencie popisov objektov vždy končí na prvom neexistujúcom čísle. Napr. ak uvedieme Object_1, Object_2, Object_4, posledný menovaný objekt už nebude braný do úvahy.

oddelených znakom „/“, pričom na konci postupnosti je názov daného listu. Ako príklad uvedieme cestu k hodnote na riadku 13: *Terrain/Object_1/Physics/2/surface*.

Na čítanie konfiguračných súborov bola implementovaná vlastná trieda – *CfgFile*. Jej použitie je veľmi jednoduché: po spustení metódy *Load* dôjde k načítaniu požadovaného súboru do internej štruktúry. V prípade potreby zistenia hodnoty niektorého z listov potom už len stačí zavolať jednu z metód *Getval_** (s cestou k listu ako parametrom). Väčšina z nich umožňuje použitie prednastavenej hodnoty, ktorá bude vrátená v prípade, že daný list neexistuje. Okrem toho je možné overiť existenciu listu pomocou metódy *Getval_exists*.

Kapitola 3 – Server / fyzikálny model

Úlohou servera v AnoRaSi je umožňovať pripájanie klientov a vytvárať simulácie, ktorých sa klienti môžu zúčastňovať a ovládať v nich jednotlivé vozidlá. Nasledujúce sekcie podávajú informácie o tom, akým spôsobom server implementovaný.

3.1 Architektúra servera

Celá funkčnosť servera je v rámci aplikácie uzavretá do objekt triedy *ServerThread*. Jediným používaným vstupným parametrom konštruktora triedy je port, na ktorom bude server očakávať pripojenia. Spustením metódy *run()* dôjde k spusteniu vlastného vlákna, čím server začne plniť svoju úlohu. V prípade potreby ukončenia jeho činnosti je možné zavolať metódu *quit()* - po návrate z nej je vlákno servera ukončené.

Vzhľadom na real-time povahu činnosti servera (simulácia prostredia) je potrebné, aby vykonávanie efektívneho kódu v ňom bolo ovplyvňované vonkajšími udalosťami – najmä sieťovou komunikáciou – čo najmenej. Preto bol celý sieťový kód presunutý do ďalšieho vlákna (definovaný triedou *ServerSocketThread*), ktoré zabezpečuje všetku komunikáciu s klientmi, vrátane spracovania ich pripájania sa. Komunikácia medzi týmito vláknami prebieha cez privátne premenné triedy *ServerThread* – *commSrvSockQueue* (správy smerujúce od servera ku klientom), *commSockSrvQueue* (opačný smer), za použitia uzamykacieho mutexu pre oba smery (*dataMutex*). Popisu formátu správ sa podrobnejšie venuje programátorská príručka a čiastočne aj 5. kapitola.

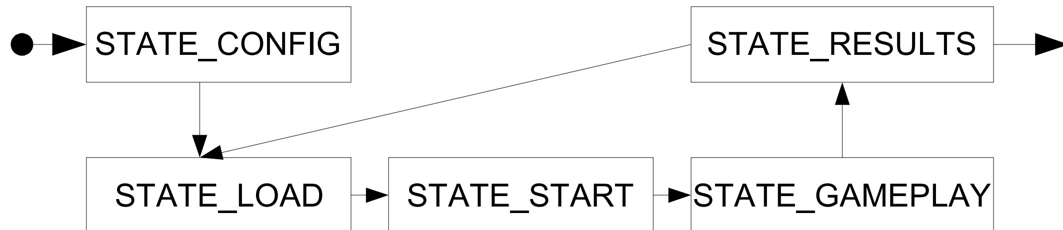
Po spustení komunikačného vlákna a inicializácii vnútorných premenných (napr. zoznamov vozidiel a máp použiteľných v simulácii) dôjde k spusteniu cyklu, v ktorom sa spracúvajú správy od klientov, vykonávajú potrebné operácie a prípadne sa správy odosielajú klientom späť. Po ukončení tohto cyklu dôjde k uvoľneniu použitej pamäte.

Server v podstate pracuje na princípe konečného automatu. Počas svojej existencie môže jeho stav (ktorá je uložený v premennej triedy *serverState*) nadobúdať nasledovných hodnôt (nasleduje zoznam hodnôt spolu so stručnými popismi, akú činnosť server v danom stave vykonáva):

- *STATE_CONFIG*: prijímanie nových pripojení; konfigurácia vlastností závodu, prijímanie požiadaviek na zmenu trate, vozidiel a nickov klientov
- *STATE_LOAD*: vytváranie objektov simulácie

- *STATE_START*: čakací stav, trvá po dobu inicializácie všetkých klientov
- *STATE_GAMEPLAY*: vykonávanie vlastnej simulácie
- *STATE_RESULTS*: zobrazenie výsledkov závodu

Tieto stavy sa môžu meniť medzi sebou len v určitom poradí, tak ako to znázorňuje nasledujúca schéma z obrázku 3.



Obrázok 3: Stavový diagram servera

Zo schémy je zrejmé, že iniciálnym stavom servera je STATE_CONFIG. Ukončiť server je síce možné v ľubovoľnom stave (šípky nie sú pre prehľadnosť zakreslené), „štandardným výstupným stavom“ je však STATE_RESULTS.

V každom zo stavov server prijíma od klientov a posiela im späť len obmedzenú množinu správ (v prípade príchodzích správ sú nespracované správy „zahodené“). Nasleduje popis komunikácie a činnosti servera v jednotlivých stavoch.

STATE_CONFIG:

Tento stav je východzím stavom servera. Server v ňom očakáva pripojenie ostatných klientov a umožňuje konfiguráciu neskôr vykonávaných simulácií. Prvou správou v komunikácii medzi odoslanou klientom serveru musí byť jedna z dvojice JOIN/SPECTATE. Zaslaním správy JOIN klient oznamuje, že sa chce zúčastniť ako hráč – že chce ovládať vozidlo. Parametrom (DATA) správy je požadovaný nick. V prípade, že klient môže byť zaradený do simulácie, je mu zaslaná naspäť správa JOIN (potvrdzuje zaradenie), LIST_CARS (zoznam dostupných vozidiel), CAR (prednastavené vozidlo pre daného klienta) a MAP (aktuálna mapa). Okrem toho server vygeneruje správu o pripojení do chatu a rozošle všetkým klientom nový zoznam pripojených. Ak na jeho zaradenie už nie je v aktuálnej mape dostatok voľných štartovacích pozícií, bude pripojený ako pozorovateľ – podobne ako keby bol poslal miesto JOIN správu SPECTATE. Sémantika tejto správy je podobná, klientovi však nie sú naspäť odosielané zoznamy možných vozidiel (ako pozorovateľ ich nebude

potrebovať). Potom, ako je klient jednou z týchto dvoch správ pripojený (a zaregistrovaný vo vnútorných štruktúrach servera), môže začať odosielať ďalšie správy. V prípade, že prvá prijatá správa je iná ako JOIN/SPECTATE, server ukončí sieťové spojenie s daným klientom.

Klienti majú právo nastaviť druh svojho vozidla (pomocou správy CAR) a nicku (NICK). V oboch prípadoch sú o úspešnej zmene klienti informovaní prijatím rovnakej správy od servera. Zároveň sú po týchto dvoch správach znovu odoslané zoznamy klientov (správa USERS).

Veľmi dôležitá je aj správa ADMIN, pomocou ktorej klient žiada o pridelenie administrátorských práv na serveri. Podmienkou pritom je, že administrátorom bude jediný klient. O získaní týchto práv je oboznámený taktiež správou ADMIN, tentokrát odoslanou serverom (tieto práva mu zostávajú počas celého spojenia so serverom). Spolu s touto správou klient dostane aj zoznam máp (LIST_MAPS). Hociktorá z nich sa potom dá nastaviť administrátorom zaslaním správy MAP s jej názvom ako parametrom (všetci klienti sú o zmene informovaní doručením rovnakej správy). Iba klient s právami administrátora má právo meniť štýl hry (GAMESTYLE), pridávať a odstraňovať tzv. botov[4] (BOT_ADD a BOT_RM) a spustiť hru (správa STARTGAME). Po prijatí správy STARTGAME dôjde (pomocou volania metódy *StartSim*) k broadcastu správ STARTGAME a SERVERINIT klientom. Prvá správa oznamuje klientom započatie simulácie, druhá to, že server bude istú dobu vykonávať načítanie dát a nebude teda odpovedať na žiadne správy. Ďalej nasleduje zmena stavu servera na:

STATE_LOAD:

Tento stav je len prechodný a je určený na vytvorenie simulácie a načítanie dát do nej (na strane servera) – to znamená vytvorenie objektu typu *SSimulation* a volanie jeho metód na inicializáciu prostredia (tým sa podrobnejšie zaoberá kapitola 3.4). Po ukončení týchto úkonov sú (pomocou metódy *SendPlayerInitInfo*) klientom odoslané informácie o simulácii. Poradie odosielaných správ je GETREADY (oznamuje klientom, že server ukončil svoju inicializáciu a teraz očakáva prípravu klienta), WORLD (určuje použitú mapu) a ďalej je odoslaných niekoľko správ CAR, ktoré popisujú vozidlá klientov použité v simulácii). Potom sú ešte broadcastované úvodné informácie o pozícii vozidiel (metódou *BroadcastPlayersInfo*) a poslednou správou pred prepnutím sa servera do nasledujúceho stavu je WAITING (tá oznamuje, že server

[4]Počítačovo ovládané vozidlá; v súčasnej verzii aplikácie je však pohyb vozidiel náhodný....

skončil s vlastnou inicializáciou a odosielaním konfiguračných správ - a možnosť inicializovať majú „teraz“ klienti).

STATE_START:

Jedinou činnosťou, ktorú server v tomto stave vykonáva je čakanie na inicializáciu klientov. Každý z klientov po jej dokončení zašle serveru správu READY (prijatie tejto správy si server zaznamená do internej štruktúry). V okamihu, keď ukončia inicializáciu všetci hráči (tzn. klienti úspešne pripojení pomocou správy JOIN) dôjde k prepnutiu stavu serveru na nasledujúci.

STATE_GAMEPLAY:

Stav, v ktorom sa je vykonávaná hlavná činnosť servera – simulácia. Táto činnosť je vykonávaná v cykle (podobne ako to robia aj ostatné stavy), jeden jeho krok je možné rozdeliť na 3 časti: spracovanie správ od klientov, vykonanie kroku simulácie, odoslanie správ späť klientom. Od klientov sú prijímané a spracovávané správy popisujúce ovládanie vozidla (CONTROL) a správy určené na zisťovanie odozvy servera (PING). Okrem toho môže administrátor poslať správu RESTART, ktorá spôsobí nové vytvorenie simulácie (nevyžaduje ale reštartovanie klientskej časti – všetky „nastavenia“ až na pozície jednotlivých objektov a nastavenia ovládania vozidiel ostávajú nezmenené). Ďalším krokom je vykonanie samotnej simulácie. Touto činnosťou sa zaoberá osobitná kapitola 3.5. Poslednou časťou cyklu je rozoslanie informácií o zmenách pozícií a ovládania vozidiel (*BroadcastPlayersInfo*) a o zmenách pozícií ostatných pohyblivých objektov (*BroadcastObjectsInfo*). Okrem toho sú po každom kroku, pomocou funkcie *BroadcastSimStatus*, odosielané aj informácie o aktuálnom čase v simulácii a o prípadných udalostiach v rámci nej (prejazd vozidiel checkpointami a podobne; tieto udalosti sú uložené v objekte simulácie SSimulation, vo fronte events). Posledne menovaná funkcia taktiež zabezpečí, v prípade ukončenia závodu riadnym spôsobom (tzn. po prejazde všetkých vozidiel potrebným počtom kôl), odoslanie výsledkov závodu klientom a prepnutie servera do nasledujúceho stavu.

STATE_RESULTS:

Tento stav je slúži na vytvorenie prestávky medzi dvomi závodmi. Dĺžku pauzy určuje znovu klient – administrátor, ktorý ju môže ukončiť odoslaním správy STARTGAME

(podobne ako to bolo v stave STATE_CONFIG). V prípade, že je možné v pretekoch pokračovať ďalším závodom (to záleží na nastavenom druhu pretekov), zmení sa stav servera znovu na STATE_LOAD. V opačnom prípade dôjde k odpojeniu všetkých klientov a k ukončeniu činnosti servera.

3.2 Úvod do knižnice ODE

AnoRaSi používa na simuláciu fyzikálneho prostredia knižnicu ODE (Open Dynamics Engine). Táto knižnica je používa dvojitú licenciu LGPL/BSD, takže je možné použiť ju v komerčných aj nekomerčných aplikáciách. Knižnica je určená na simuláciu dynamiky spojených tuhých telies[5]. Je optimalizovaná na vykonávanie simulácií pracujúcich v reálnom čase. Okrem toho umožňuje zmenu prostredia počas simulácie a je tak vhodná na použitie pri vytváraní prostredí virtuálnej reality.

Na výpočty používa vysoko stabilný integrátor, takže chyby simulácií by sa nemali vymykať spod kontroly. Fyzikálne sa tým myslí to, že simulovaný systém by nemal bezdôvodne „explodovať", ako sa to stáva u niektorých iných knižníc. ODE preferuje rýchlosť a stabilitu pred fyzikálnou presnosťou. Používa tzv. „tvrdé kontakty", čo v praxi znamená že pri kolízii sú telesá prinútené neprenikať do seba navzájom. Okrem toho knižnica disponuje vlastným systémom detekcie kolízií, ktorý podporuje telesá ako guľa, hranol, zaguľatený valec, jednoduchá plocha či trojuholníková sieť. Tento systém je rýchly aj vďaka použitiu konceptu „space"-ov, ktoré umožňujú vytvoriť hierarchickú štruktúru objektov v simulácii.

Cieľom tejto práce nie je nahradiť dokumentáciu ODE, preto tu budú vysvetlené len základné vlastnosti knižnice, úplnejšie informácie ctený čitateľ nájde v [2]. Informácie použité v nasledujúcom oddieli čiastočne pochádzajú z dokumentácie knižnice.

3.3 Popis objektov a ich vlastností v ODE

Základným prvkom ODE simulácie sú pevné telesá. Každé z telies má niekoľko dynamických vlastností: jednou skupinou sú také, ktoré sa počas simulácie menia (pozícia a lineárna rýchlosť pohybu referenčného bodu telesa; rotácia telesa a uhlová rýchlosť jeho otáčania), druhou vlastnosti ostávajúce počas simulácie spravidla konštantné (hmotnosť telesa a distribúcia hmoty v jeho vnútri). Ako si mohol čitateľ

[5]Pod pojmom spojené tuhé telesá sa myslí množina telies, ktoré sú prepojené nejakým druhom kĺbu, behúňom a pod.

všimnúť, nikde sa tu nespomína tvar telesa. ODE odlišuje dynamické vlastnosti od tvaru telesa – ten sa totiž využíva „iba“ na detekciu kolízií.

Simulácia pomocou ODE prebieha po krokoch (časových intervaloch), pričom dĺžku intervalu si môže užívateľ určiť. Presnosť integrácie vykonanej vo vnútri knižnice záleží nepriamo úmerne na veľkosti časového intervalu, cez ktorý sa integruje – nikdy však nie je dokonale presná. Napriek tomu jej výsledky väčšinou vyzerajú dostatočne realisticky.

Aby mohol čitateľ konfrontovať informácie v nasledujúcich odsekoch s dokumentáciou ODE, budeme čiastočne používať anglické výrazy – joint (spoj), hinge (pánt, kĺb). V skutočnom svete je joint niečo ako pánt, ktorý je použitý na spojenie dvoch objektov. V ODE je joint realite veľmi podobný: je to vzťah, ktorý núti dva telesá nachádzať sa iba v relatívnych pozíciách a orientáciách, ktoré daný druh joint-u umožňuje. Tento vzťah sa nazýva obmedzenie. Pritom pojmy joint/spoj a obmedzenie sú v ODE považované za ekvivalentné. V každom kroku simulácie je umožnené každému jointu aplikovať obmedzujúce sily. Tieto sily potom zaručujú, že jednotlivé pevné telesá ostanú v správnej relatívnej pozícii.

Problém s nepresnosťou integrácie a taktiež s nepresnosťou implementácie desatinných čísel sa však prejaví už v tomto momente – je zrejmé, že žiadnym spôsobom nie je možné udržať jednotlivé telesá na takých pozíciách, aby presne vyhovovali obmedzeniam jointov. V najlepšom prípade sa môže stať (a tento prípad nastáva takmer vždy) že dôjde k nesprávnemu zaokrúhleniu a telesá sú posunuté do zlej pozície. V takýchto prípadoch je uplatnený mechanizmus na kompenzáciu týchto chýb. Jednou časťou tohto mechanizmu je tzv. ERP[6]. Je to hodnota v rozsahu <0, 1>, ktorá určuje ako veľmi sa bude simulácia v každom kroku „vyrovnávať“ pozície objektov tak, aby boli zachované obmedzenia vyplývajúce z prepojení pomocou jointov. Pritom 0 znamená nepoužívanie opravujúcich síl, hodnoty blízke 1 naopak maximálne vyrovnávanie (to však nie je odporúčané vzhľadom na vnútorné aproximácie). Druhá časť mechanizmu je CFM[7]. Vysvetlenie je zložitejšie, ale vo väčšine prípadov by táto hodnota mala byť nezáporná. Ak je to 0, bude daný spoj „tvrdý“ - nebude tolerovať žiadne odchýlky pozícií. V praxi sa to však dá kompenzovať práve pomocou síl vyplývajúcich z použitia parametra ERP. Ak bude hodnota väčšia ako 0, bude spoj

[6]Error Reductin Parameter

[7]Constraint Force Mixing

tolerovať odchýlky proporcionálne k veľkosti parametru CFM krát sila potrebná na návrat do správnej pozície. Pomocou týchto dvoch parametrov sa potom dajú simulovať tak spoje typu špongia (ktorá iba brzdí vzájomný pohyb telies) aj pružina (vďaka ktorej by dva telesá mohli oscilovať).

V praxi však je však takýto spôsob nastavovania vlastností spojov dosť neintuitívny a je vhodnejšie použiť vzťah na ich výpočet z konštánt k_d (damping, tlmenie) a k_p (spring, pružnosť):

$$ERP = h.k_p / (h.k_p + k_d)$$

$$CFM = 1 / (h.k_p + k_d)$$

Parameter h je dĺžka kroku simulácie – preto je potrebné pred každým z nich znovu nastaviť parametre ERP a CFM jointov v simulácii. Tieto vzťahy v končenom dôsledku umožňujú vytvorenia rovnakého efektu spoja, ako má systém pružiny a tlmiča – simulovaný implicitnou integráciou prvého rádu (viď [2], oddiel 3.8.2).

Ako už bolo spomenuté, telesá môžu byť rôznych tvarov (guľa, kváder, trojuholníková sieť atď.). Každému z týchto objektov (nazývaných „geom“) je možné priradiť tzv. „body“, reprezentujúci práve už spomenuté dynamické vlastnosti telesa. V prípade, že ku geom objektu nie je priradený body, je geom statický a nie je možné pohnúť nim žiadnou silou. Okrem toho je možné každému geom objektu priradiť ľubovolnú hodnotu (ODE ju žiadnym spôsobom nevyužíva, je určená pre programátorov na uloženie vlastných dodatočných informácií o objekte).

U telies s priradeným body je zrejmé, že vzhľadom k ich pohyblivosti počas simulácie dochádza nevyhnutne k vzájomným kolíziám (samozrejme môže pohyblivé teleso naraziť aj do nepohyblivého). ODE na ich detekciu poskytuje algoritmy, používateľ však môže použiť vlastné. A je to práve používateľ knižnice, kto musí zabezpečiť správnu reakciu pri kolízii. Táto reakcia spočíva vo vytvorení dočasných jointov medzi jednotlivými telesami pred vykonaním vlastného kroku simulácie. Jointy zabezpečia, že telesá cez seba nebudú prenikať (samozrejme, pokiaľ to nebude žiaduce) – a pomocou týchto jointov sa dajú zabezpečiť aj rôzne vlastnosti kontaktu plôch: klzavosť, pružnosť či pohlcovanie nárazov. Dočasné jointy by mali byť po každom

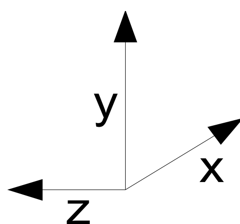
kroku zo simulácie odstránené (a eventuálne neskôr, v prípade potreby, znovu vytvorené).

Objektom ODE, ktorý celú simuláciu (tuhé telesá a jointy) spravuje je objekt „world“. Pomocou neho je možné nastaviť vektor, ktorým bude pôsobiť gravitácia na všetky telesá simulácie (funkcia dWorldSetGravity) a napríklad aj globálne hodnoty ERP a CFM.

3.4 Vytvorenie realistickej simulácie

ODE ako taká nenúti používateľov/programátorov k používaniu štandardných veličín ako sú sekundy, kilogramy či metre – dokonca nevnucuje ani použite tiažového zrýchlenia. Táto voľnosť použitia síce rozširuje možnosti knižnice, no v prípade nesprávneho použitia by mohla viesť k značne nerealistickému správaniu telies v simulácii. Preto bolo už od začiatku vytvárania simulácie dôsledne používať správne fyzikálne veličiny.

Jedným z najdôležitejších krokov pri vytváraní simulácie je zvolenie súradnicového systému. Kvôli čím väčšej kompatibilite s klientskou časťou sme použili na oboch „stranách“ trochu nezvykle orientovaný ľavotočivý kartézsky súradnicový systém (šípky ukazujú kladný smer), zobrazený na obrázku 4.



Obrázok 4: Použitá súradnicová sústava

Na zvislej súradnici bude pôsobiť simulované gravitačné zrýchlenie. Pre zachovanie istej voľnosti pri tvorbe nových tratí je možné pre každú z nich určiť vlastné – konkrétne v položke jej konfigurácie *Physics/gravity*. Vo väčšine prípadov to bude záporná hodnota (a objekty budú teda urýchľované smerom nadol) – pre dosiahnutie štandardných podmienok $-9.81\ ms^{-2}$. Pomocou funkcie ODE – dWorldSetGravity – sa potom nastaví vektor zrýchlenia na (0, gravity, 0).

Druhým krokom je vytvorenie prostredia obklopujúceho vozidlá. To sa vytvára podľa administrátorom zvolenej trate. Každý z objektov prostredia je popísaný v jej konfiguračnom súbore v podstrome *Terrain/Object_N*, kde N je poradové číslo objektu. Položke *Terrain/Object_N/type* určuje typ výsledného objektu. Vzhľadom na relatívnu jednoduchosť implementácie sú programom podporované 3 druhy objektov – kvádre (hodnota *box*), gule (*sphere*) a objekty tvorené trojuholníkovou sieťou (*object*)[8]. Ostatné typy sú serverovou časťou aplikácie ignorované. Tak isto sú ignorované objekty, ktoré obsahujú hodnotu *skip=yes*. Nutnou podmienkou zaradenia objektu do simulácie je aj špecifikovanie jeho povrchu (pre kvádre a gule sa určuje v podpoložke objektu *Physics/surface*; trimesh objekty môžu mať viacero povrchov – viď ďalej) - v opačnom prípade bude objekt iba zobrazený v klientskej časti a nebude mať žiadny vplyv na simuláciu.

Kvádre sú popísané pomocou rozmerov (hodnota *scale*), pozície (*pos*) a rotácie (*rot*). Podobne guľa má polomer (*radius*), pozíciu a rotáciu (tá síce nie je pre simuláciu podstatná, pre zobrazenie telesa je ale dôležitá). V prípade, že je telesu priradená v konfigurácii aj hmotnosť (*Physics/weight*), bude s „geom“ daného telesa asociovaný aj body – bude ním teda možné v simulácii pohybovať (nárazmi, pôsobí naň gravitácia a pod.)

Špeciálne vlastnosti majú trimesh telesá. Pri jeho vytváraní je potrebné v položke *Object_N/file* určiť súbor, z ktorého bude načítaný jeho model, Každý trimesh objekt sa môže skladať s viacerých podmodelov. Kvôli tomu, a kvôli zložitosti určovania ťažiska nie je možné trimesh objektu priradiť hmotnosť a vždy je statický. Okrem povrchu je možné určiť len jeho mierku (*Object_N/scale*), a teda nie pozíciu ani rotáciu (to je však možné čiastočne obísť premiestnením vrcholov už v modelovacom programe, v ktorom sa daný trimesh vytvára). Povinnosťou autora mapy je, ak vyžaduje zaradenie objektu do simulácie, špecifikovať povrchy pre všetky jeho časti. Pre každú z častí sa povrch určí v položke *Physics/N/surface* (kde N je číslo povrchu).

V predchádzajúcich odstavcoch bolo spomenuté, že telesám je možné nastaviť povrch. Jednotlivé povrchy sú definované v súbore *data/settings.ini* v položkách

[8]Pridať ďalšie druhy objektov by bola viac menej triviálna mechanická „práca“ - dôležité len je, aby fyzikálne vlastnosti telesa (poloha, rozmery) v simulácii zodpovedali zobrazeniu na klientskej strane (ktoré by muselo byť taktiež doprogramované).

Physics/Surface_N. Pritom N je index povrchu a položka *name* určuje meno povrchu (ktoré sa pre väčšiu intuitívnosť používa v konfiguračných súboroch miesto indexu). Pre každú dvojicu povrchov je ďalej možné špecifikovať vlastnosti ich kontaktu (tieto informácie sú využité až počas simulácie, konkrétne pri detekcii kolízií a vytváraní jointov). V podstrome *Physics/SurfaceContactProperties* sa nachádza množina hodnôt s kľúčmi tvaru „A+B=mu slip damp spring", pričom musí platiť A<B. Takáto hodnota označuje vlastnosti kontaktu medzi 2 objektmi s povrchmi s indexmi A a B[9]. V prípade, že pre danú dvojicu objektov nie je sú explicitne špecifikované vlastnosti, budú použité vlastnosti v *Physics/SurfaceContactProperties/default.*

Pre samotnú hru je potrebné do prostredia umiestniť určité objekty, checkpointy a finish (súhrnne označené ako gameplace objekty), ktoré síce nebudú nijak fyzikálne interagovať s ostatnými súčasťami simulácie, neskôr však umožnia jednoduché počítanie kôl prejdených vozidlami. V AnoRaSi majú oba druhy objektov rovnaký tvar – kváder – a tak je možné ich vytvoriť podobným spôsobom ako „skutočné" kvádre. Checkpointy sú definované v položkách *Checkpoints/Check_N*, finish v *Checkpoints/Finish*. Parametrami sú intuitívne *pos*, *rot* a *size*.

Posledným a najzložitejším krokom pri vytváraní simulácie je inicializácia vozidiel. Každé z nich sa do simulácie vkladá osobitne – nezávisle na tom, či sú niektoré z nich rovnakého typu. Pre každé vozidlo sa načíta súbor s konfiguráciou a podľa nej sa postupne do simulácie vkladajú a spájajú jednotlivé súčasti. Jednotlivé vozidlá v simulácii sú logicky vytvárané na rôznych miestach, preto sa pri definícii pozícií jednotlivých objektov v konfiguračnom súbore používa súradnicová sústava s počiatkom v (0, 0, 0). Pri vkladaní prvkov vozidla do simulácie si program sám prepočíta súradnice objektov vo virtuálnom svete.

Dôležitou súčasťou vozidla je jeho karoséria. Vzhľadom na to, že knižnica ODE dokáže pracovať priamo s trimesh objektmi, bolo by v princípe jednoduché vytvoriť karosériu priamo z modelu. Problémom však je výpočtová zložitosť – trimesh sú zďaleka najzložitejšie objekty, čo sa týka zisťovania kolízií s ostatnými telesami. Použitie modelu by mohlo prichádzať do úvahy v prípade, že by sa dostatočne zredukovala trojuholníková sieť modelu. Toto riešenie je však v konflikte s tým, že by sme daný model použili aj v klientskej časti – kde je naopak snaha o čo najviac detailov.

---

[9]Na poradí samozrejme v tomto prípade nezáleží – z hľadiska vlastností kontaktu (ako je šmyklavosť, tlmenie nárazov) je napríklad jedno, či je ťahané gumené teleso po kove, alebo kovové teleso po gume.

Ďalšia nevýhoda takéhoto riešenia je v podstate nemožnosť intuitívne určiť rozloženie hmoty vo vozidle. Na definíciu fyzického tvaru karosérie bol teda zvolený iný prístup: celá je vytvorená pomocou vhodne umiestnených a otočených kvádrov. Definované sú v prvkoch *Chassis/N/*, a to pomocou *pos*, *scale* a *rot* (význam týchto hodnôt je rovnaký ako pri vyššie spomenutom vytváraní objektov typu box). Okrem toho je možné pre každú súčasť špecifikovať hmotnosť a tak ovplyvniť polohu ťažiska výsledného telesa. V prípade neurčenia hmotnosti bude telesu nastavená hmotnosť v rádoch gramov – aby čím menej ovplyvňovalo dynamické vlastnosti vozidla. Výsledné teleso karosérie je tvorené viacerými (v rámci vozidla pevne umiestnenými) geom objektami a jedným body objektom, ktorý určuje rýchlosť, zotrvačnosť karosérie a taktiež rozloženie hmoty okolo ťažiska.

Kolesá umožňujú vozidlu pohyb vďaka okrúhlemu tvaru. ODE však v podstate neobsahuje žiadny geom tvaru valca. Preto bolo potrebné použiť náhradu - aproximáciu guľou. Na druhej strane ODE obsahuje body „tvaru" valca. Spojením týchto dvoch súčastí tak síce dostaneme objekt, ktorý má mierne väčší objem[10], ako by malo mať skutočné koleso, zato však má rovnaké dynamické vlastnosti (zotrvačnosť a pod.). Pre zjednodušenie nastavenia kolies je možné do konfigurácie vložiť položku *Body/Wheel* – v nej sa budú sa hľadať nastavenia, ak sa v konfigurácii jednotlivých kolies neurčí niektorá z vlastností (to samozrejme neplatí pre všetky položky, napr. by bolo zbytočné mať prednastavenú pozíciu kolesa). Okrem štandardných vlastností ako je polomer (*radius*), šírka (*width*), hmotnosť (*weight*) a pozícia kolesa (stále vzhľadom na počiatok súradnicovej sústavy) je možné určiť aj typ kolesa a vlastnosti závesu (spoja medzi kolesom a karosériou). Položka (súbor atribútov) *attr* slúži na určenie, či bude koleso udržovať jeden smer (hodnota *STRAIGHT*) alebo či bude umožňovať otáčanie okolo zvislej osi (*STEER*) – v tom prípade je možné určiť, že sa koleso bude otáčať v opačnom smere ako požaduje hráč (*REVERSED*). Ďalej môže ľubovoľné koleso slúžiť na využitie sily motora k pohybu vozidla (*THURST*). Vlastnosti závesu sú určené konštantami *damp* a *spring*, ktorých význam už bol popísaný v sekcii 3.2. Poslednou nastaviteľnou hodnotou je *brakes*, ktorou sa neskôr násobí brzdiaca sila (a teda je možné túto položku použiť napríklad na rôzne rozloženie brzdného účinku na predné a zadné kolesá, viď kapitola 3.6). Každé z kolies je po vytvorení príslušného geom a body objektu pripojené ku karosérii pomocou jointu Hinge2, ktorý v podstate predstavuje štandardné nezávislé zavesenie kolesa vo vozidlách (s určitým

[10]To sa prejavuje napr. pri kolíziách s terénom a okolitými vozidlami.

zjednodušením). Jointu sú nastavené obmedzenia tak, aby s neumožňoval otáčanie kolesa v prípade, že sa jedná o „straight koleso“. Po vytvorení všetkých vozidiel je simulácia pripravená na používanie serverom.

3.5 Jeden krok simulácie

Udržiavanie simulácie v AnoRaSi pozostáva z viacerých úkonov a v podstate sa drží postupu uvedeného v [2], odseku 3.10. V nasledujúcich odstavcoch sú jednotlivé úkony popísané podrobnejšie.

Pre každý z krokov je potrebné určiť, aký dlhý časový úsek sa bude simulovať. Ako už bolo napísané, presnosť simulácie závisí nepriamo úmerne veľkosti úseku. Dĺžku časového úseku však nemôžme zmenšovať neobmedzene – čím menší ho zvolíme, tým viac krát bude potrebné celú procedúru úkonov opakovať na simuláciu napríklad jednej sekundy. Tým samozrejme rastú aj nároky na výpočtový výkon procesora. Je zrejmé, že pri istej veľkosti kroku by sa už mohlo stať, že simuláciu nebude možné vykonávať realtime – čas v simulácii bude „plynúť pomalšie“ ako by v skutočnosti mal. Okrem toho sú hneď po vykonaní kroku simulácie odosielané dáta o pozíciách klientom – mohlo by tak dôjsť k zahlteniu komunikačných kanálov. Aby sme sa vyhli podobným problémom, určujeme dĺžku intervalu dynamicky – odčítaním času posledného kroku simulácie od aktuálneho. Minimálna dĺžka jedného kroku je určená na 0.01s – kratšie kroky nie sú vykonané. Dĺžka je obmedzená aj zhora – kroky dlhšie ako 0.05s sú simulované ako keby trvali práve 0.05s. Obmedzenie zhora sa stáva väčšinou na pomalých počítačoch – dôjde tým síce k spomaleniu simulácie, z veľkej časti však zabránime fatálnym chybám výpočtov (najčastejšie sa prejavujúcich nesprávnou detekciou kolízií).

Prvým úkonom je nastavenie ovládania každého z vozidiel (podľa „želaní vodičov“) a aplikácia síl na ne pôsobiacich. Fyzike vozidla sa podrobnejšie venuje nasledujúca sekcia.

Druhým úkonom je detekcia kolízií v simulácii. Tá sa vykoná volaním funkcie dSpaceCollide (resp. metódy collide objektu dSpace), ktorej parametrom je okrem toplevel space objektu simulácie aj odkaz na callback funkciu (v prípade AnoRaSi sa používa statická metóda triedy *SSimulation::nearCallback*) a vlastný parameter, ktorý

bude tejto funkcii dodaný (ten nastavujeme na pointer na *SSimulation* objekt, reprezentujúci aktuálnu simulácia). Callback funkcia je volaná vždy po detekovaní možnej kolízie dvoch objektov. Hlavnou úlohou tejto funkcie je vytvárať v prípade kolízie (a potreby) dočasný joint, ktorý zabráni prieniku telies cez seba.

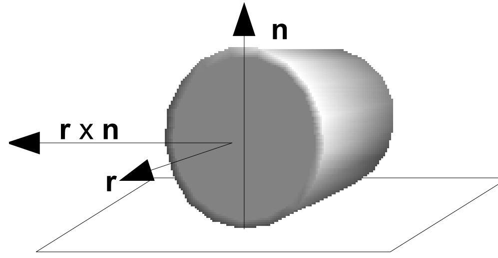
Vstupným parametrom funkcie *nearCallback* sú 2 ID kolidujúcich geom resp. space objektov a hodnota zadaná funkcii dSpaceCollide (viď predchádzajúci odstavec). ODE používa hierarchickú štruktúru space-ov, a tak v prípade že jeden z kolidujúcich objektov je typu space, prvá časť funkcie zabezpečí rekurzívne volanie dSpaceCollide. Druhá časť sa už venuje samotnému procesu vyhodnocovania kolízií medzi geom objektami. Ku každému objektu (pohyblivému aj statickému) v simulácii AnoRaSi je priradený odkaz na štruktúru odvodený od triedy *SGeomObj*. Okrem iného je priamo v tejto triede uvedené, akého typu daný objekt je - môže to byť jeden z typov *TERRAIN* (objekt/časť základného prostredia simulácie), *CAR_CHASSIS* (karoséria vozidla), *CAR_WHEEL* (koleso vozidla) a *GAMEPLACE* (checkpoint/finish).

Prvou testovanou dvojicou objektov, medzi ktorými môže dôjsť ku kolízii sú karoséria vozidla a objekt gameplace. Dôležitou vlastnosťou tejto kolízie je, že nie sú vytvárané žiadne jointy (na rozdiel od ostatných situácií), ktoré by zabraňovali vozidlu prejsť cez daný objekt. Spôsob využitia týchto kolízií je popísaný v sekcii 3.7.

Z ďalších testov sú potom postupne vylúčené dvojice objektov gameplace; objekty, ktoré už sú spojené nejakým typom jointu a konečne objekty patriace do jedného vozidla (tak zabránime prípadným kolíziám kolies vozidla s karosériou, ktoré by mohli nastávať pri nevhodnom rozmiestnení prvkov karosérie).

Všetky zvyšné kolízie sú ďalej riešené vytváraním (dočasných) jointov, ktoré automaticky vytvára funkcia ODE dCollide. Po ich vytvorení je potrebné určiť ich vlastnosti – a tým aj vlastnosti kontaktov povrchov jednotlivých dvojíc telies. V prípade, že sa ani jedno z telies nie je kolesom, je toto určenie implementované náhľadom do tabuľky, ktorá bola vytvorená počas inicializácie servera zo súboru *data/settings.ini*. Kontakt kolesa s ostatnými telesami je pre väčšiu realistickosť spracovaný ako špeciálny prípad. Vlastnosťou kolies je totiž to, že zväčšovaním rýchlosti ich otáčania dochádza k zmenšovaniu odporu voči pohybu v smere kolmom na pohyb kolesa. V praxi to znamená, že pri väčšej rýchlosti otáčania kolies sa koleso ľahšie dostane do bočného šmyku, kým ak je koleso v pokoji, je na jeho bočný pohyb potrebná oveľa väčšia sila. Preto pre každý kontakt kolesa s povrchom je pre potreby ODE potrebné určiť smer pohybu kolesa. Ten sa vypočíta z vektoru jeho rotácie $\vec{r}$ a z

normálového vektoru $\vec{n}$ kontaktu plôch - vektorovým súčinom, ako je to zjednodušene znázornené na nasledujúcej schéme v obrázku 5.



Obrázok 5: Výpočet vektoru pohybu

Šmykľavosť (atribút ODE kontaktu slip1) v smere otáčania kolesa je potom nastavená rovnako, ako je „bežná“ šmykľavosť medzi materiálmi kolesa a daného objektu; v kolmom smere (slip2) je potom priamo úmerná (jednej desatine) rýchlosti pohybu kolesa. Okrem toho je potrebné nastaviť aj atribút fdir1 – už vypočítaný vektor smeru otáčania kolesa. Pre oba druhy kontaktov (s účasťou kolies aj bez nich) je potom ešte potrebné nastaviť parametre ERP a CFM, ktoré sa prepočítajú z dĺžky ďalšieho kroku simulácie a vlastností kontaktu damp a spring, pomocou vzorca uvedeného v kapitole 3.3.

Po vytvorení kontaktov dôjde k vykonaniu simulácie pomocou metódy objektu dWorld::step (s už vypočítanou dĺžkou časového úseku ako parametrom). Návratom z tejto metódy a vymazaním vytvorených dočasných jointov je krok simulácie ukončený.

3.6 Fyzika automobilu

Vozidlo v AnoRaSi je tvorené, ako už bolo napísané, z karosérie a kolies. Nie je samozrejme možné simulovať celé vozidlo dokonale – teda aplikovať všetky pôsobiace sily. Do AnoRaSi boli teda vybrané len sily, ktoré sa najvýznamnejším spôsobom podieľajú na vlastnostiach a ovládaní vozidla. Okrem tých, ktoré sú vytvárané priamo knižnicou ODE[11], sú ostatné sily pôsobiace na vozidlo popísané vo zvyšku tejto sekcie.

Základným spôsobom ovládania vozidla je otáčanie kolesami. Napriek tomu, ako triviálne táto činnosť pôsobí, je vcelku zložité určiť silu, ktorú potrebujeme na otočenie kolesom. Kvôli realistickosti nie je vhodné ani jednoducho nastaviť otočenie kolesa.

[11] Napríklad sily spôsobujúce tlmiace a pružinové vlastnosti jointov medzi kolesami a karosériou.

Ďalším problémom v ODE je, že na nastavenie otočenia kolesa nemôžme, kvôli nestabilite simulácie, použiť ľubovoľne veľké sily. Na dosiahnutie otočenia kolesa sa teda používa trik: Pre daný joint nastavíme maximálne uhly otočenia kolesa (parametre dParamHiStop a dParamLoStop) a maximálnu silu použiteľnú na dosiahnutie otočenia (dParamFMax, nastavuje sa na 1000 N). Miesto priameho nastavenia vyžadovaného otočenia kolesa do parametra dParamVel ale nastavujeme niekoľko krát väčšiu hodnotu (závislú na aktuálnom a požadovanom otočení) a k požadovanému otočeniu kolesa konvergujeme postupne. Tento postup funguje a napriek tomu že nie je úplne korektný, poskytuje vcelku realistické správanie kolies, čo sa týka ich otáčania okolo zvislej osi.

Pohyb auta je možný najmä vďaka prenosu sily motora na kolesá. Motor ako hnacia jednotka má 2 základné vlastnosti: rozsah pracovných otáčok a krivka točivého mementu, ktorý je schopný motor dodať pri plnom výkone pri daných otáčkach. Obe vlastnosti sú pre zjednodušenie určené v konfigurácii vozidla vo 4 dvojiciach položiek *Dynamics/Engine/torque_Nx* a *Dynamics/Engine/torque_Ny.* Tieto dvojice definujú beziérovu krivku (tá by mala byť, pre správne fungovanie výpočtov, 2D funkciou závislou na x), pričom x-ová súradnica jej prvého a posledného bodu určuje minimálne, resp. maximálne otáčky motora. Pre výpočet točivého momentu pri daných otáčkach sa vnútorne najprv otáčky normalizujú na hodnotu v rozsahu <0, 1> a y-ová súradnica krivky v danom bode je hľadaný moment. Motor samozrejme pri daných nedodáva maximálny točivý moment vždy – jeho veľkosť záleží aj na polohe „plynového pedálu". Situácia je o to komplikovanejšia, že v prípade nedostatočného prísunu paliva začne motor dodávať opačný točivý moment a dochádza k znižovaniu rýchlosti motorom. Ďalšou súčasťou pohonného systému vozidla je prevodová jednotka. Tá sa stará hlavne o prenos a o, pomocou rôznych prevodov, vhodné využitie točivého momentu na pohon kolies. Prevodovka je s motorom prepojená pomocou spojky, ktorá v realite zabezpečuje plynulé zmeny prevodov a ďalšie pre túto simuláciu nepodstatné činnosti. Posledným článkom prevodového ústrojenstva medzi motorom a kolesami je diferenciál. Jej skutočným účelom je zamedzovať prílišnému opotrebeniu pneumatík na hnacích kolesách, okrem toho však zabezpečuje aj ďalšiu (tentokrát konštantnú) zmenu pomeru otáčok motora a kolies – čo sa samozrejme musí v simulácii prejaviť.

Ani práca motorového ústrojenstva v AnoRaSi nie je simulovaná dokonale. Otáčky motora sú pre jednoduchosť počítané z otáčok kolies (za použitia vhodného koeficientu, daného prevodovým stupňom a podobne). Pri tomto postupe sa však môže stať, že

otáčky motora sa dostanú mimo jeho pracovný rozsah (pri rozbiehaní by boli príliš nízke či nulové; po nevhodnom podradení by sa zase mohli dostať vysoko nad maximálne otáčky). Riešenie spočíva v obmedzení otáčok zdola, respektívne zhora na pracovný rozsah – vďaka tomuto sa výsledné ovládanie podobá tomu, ako keby pri bola nevhodných otáčkach automaticky stláčaná spojka.

V prípade, že vyžadované otáčky motora (ktoré sa je možné spočítať z polohy plynového pedálu) sú väčšie ako aktuálne, vypočíta sa výsledný točivý moment dodaný motorom kolesám ako

$$T = maxTorque(currentRpm) * gR * tEff * (1 - clutch) * accel$$

kde *maxTorque(currentRpm)* je maximálny možný točivý moment dodávaný motorom pri daných otáčkach, *gR* je keoficient aktuálneho prevodového stupňa, *tEff* účinnosť prenosu energie medzi motorom a kolesami, *clutch* sila stlačenia spojky (v rozsahu <0,1>, kde 0 znamená nestlačenú spojku) a *accel* je sila stlačenia plynu (taktiež hodnota <0,1>).

Ak nastane opačný prípad – vyžadované otáčky motora sú nižšie ako aktuálne, dochádza k brzdeniu motorom. Motor potom dodáva kolesám točivý moment opačný smeru otáčania kolies. Jeho veľkosť vypočítame rovnako ako v predchádzajúcom prípade, ale je ešte vynásobený konštantou z konfigurácie vozidla – *Dynamics/Engine/breaking_power*[12]. Vypočítaný točivý moment je následne vydelený počtom hnacích kolies pomocou funkcií ODE aplikovaný na každé z nich.

Posledným ovládacím prvkom vozidla sú brzdy, pričom existujú 2 druhy – „nožná" a ručná. Účinok v realite závisí na viacerých veciach – na sile stlačenia brzdového pedála (resp. „zatiahnutia" ručnej brzdy), rýchlosti rotácie kolies, opotrebení brzdných doštičiek, ich teplote atď. Pre účely simulácie sa obmedzíme len na prvú vlastnosť. Spomaľovací účinok bŕzd je daný dodávaním opačného točivého momentu kolesám, ako je smer ich rotácie. Problém v simulácii ale je dodanie toho správneho točivého momentu tak, aby nedošlo k zmene smeru otáčania kolies (čo by bolo zrejme v hrubom rozpore s realitou). Preto je použitý alternatívny prístup – ODE umožňuje priamo nastavenie otáčok kolesa (v tomto prípade sa budeme snažiť o dosiahnutie nulových otáčok – zastavenie) a maximálnu silu, ktorú je k tomu možné použiť. Vďaka tomuto

[12]Táto konštanta určuje aký veľký točivý moment dokáže motor dodať pri voľnobehu na daných otáčkach. Použitie konštanty je na tomto mieste čisto empirické, keďže autor nikde nenašiel konkrétne vzťahy pre takéto prípady.

postupu nedôjde k spomínanému neželanému „pretáčaniu kolies" a znovu – efekt bude vcelku realistický. Ak je zatiahnutá ručná brzda, výsledná brzdná sila pôsobiaca na každé z kolies je

$$F_{break} = HandbrakeForce_{max} * handbrake$$

kde $HandbrakeForce_{max}$ je maximálna sila (N) ručnej brzdy (parameter *Dynamics/Brakes/max_handbrakeforce*) a handbrake je poloha jej zatiahnutia (<0, 1>, 0 znamená uvoľnenú brzdu). V prípade, že je stlačený pedál „nožnej brzdy", brzdnú silu vypočítame obdobne:

$$F_{break} = BrakeForce_{max} * brake * ratio$$

kde, $BrakeForce_{max}$ je maximálna sila (N) brzdy (parameter *Dynamics/Brakes/max_brakeforce*) a brake je poloha stlačenia pedála (<0, 1>, 0 znamená uvoľnenú brzdu). Koeficient ratio umožňuje rozloženie brzdného účinku rovnomerne medzi predné a zadné kolesá (každému kolesu je možné v konfigurácii vozidla určiť iný koeficient pomocou atribútu *brakes*).

Silou aplikovanou na vozidlo, ktorá nemá v realite príliš obdobu (snáď s výnimkou použitia žeriavu) je aplikovanie sily (v smere opačnom smeru gravitačnej sily) na roh vozidla. Táto sila umožňuje v prípade prevrhnutia vozidla na strechu jeho otočenie do správnej polohy. Jej veľkosť je daná ako násobok celkovej tiaže karosérie vozidla.

Po nastavení ovládania vozidla je potrebné nastaviť ešte sily, ktoré ale neovplyvňuje hráč priamo. Na karosériu pôsobí, okrem iného, brzdnou silou okolitý vzduch – s rýchlosťou rastie jeho odpor kvadraticky. Táto sila pôsobí v smere opačnom pohybu voči vozidla, a jej veľkosť je

$$\vec{F}_{aero} = -C_{drag} * \vec{v} * |v|$$

kde C_{drag} je konštanta závislá na okolitom prostredí a tvare karosérie (a je zadaná priamo v konfigurácii vozidla) a $\vec{v}$ je jej pohybový vektor.

Pri pohybe kolies vzniká trenie lineárne závislé na rýchlosti ich pohybu – tzv. valivý odpor. Simulujeme ho pomocou dodávaného záporného točivého momentu kolesám. Veľkosť momentu pre každé z kolies počítame podľa nasledujúceho vzorca:

$$T_{rr} = radius_{wheel} * rollResistance * angleRate * radius_{wheel} = radius_{wheel} * F_{rr}$$

$radius_{wheel}$ je polomer daného kolesa, rollResistance je konštanta jeho valivého odporu (*Dynamics/Tires/rolling_resistance*) a angleRate je uhlová rýchlosť otáčania. Posledné tri súčinitele pritom udávajú silu aplikovanú v mieste kontaktu kolesa a povrchu.

Vzťahy použité v tejto sekcii boli do značnej miery prevzaté z webového dokumentu [3] od Marca Monstera, ktorý však už nebol v dobe písania práce dostupný. Preto je jeho kópia umiestnená na pridanom médiu.

3.7 Hra

Okrem simulácie sveta umožňuje AnoRaSi závodenie hráčov. Na strane servera sú podstatné len informácie o poradí vozidiel hráčov v simulácii. V štruktúrach o jednotlivých vozidlách sú uložené zoznamy prejdených checkpointov a počty vykonaných kôl. Na začiatku simulácie (tzn. aj po jej reštarte) je počet kôl vozidla nastavený na 0, a všetky checkpointy označené za neprejdené. Akým spôsobom sa zisťuje prejdenie vozidla checkpointom (resp. finishom) bolo naznačené v kapitole 3.5.

Pre každú z tratí je možné v jej konfigurácii určiť poradie (*Checkpoints/order*), v ktorom je potrebné jednotlivé checkpointy prejsť – *random* alebo *sequential*. Random mód znamená, že jednotlivými checkpointami je možné prejsť v ľubovoľnom poradí. Naproti tomu sequential označí checkpoint za prejdený, iba ak vozidlo prešlo všetkými ostatnými checkpointami s nižším indexom (prvý checkpoint je samozrejme možné prejsť hneď po štarte). Keď vozidlo prejde všetkými checkpointami (vo vhodnom poradí), je možné uzavrieť kolo prejdením cez finish – čím dôjde k zvýšeniu počtu prejdených kôl.

Potom, ako všetky vozidlá prejdú administrátorom stanovený počet kôl, dôjde k ukončeniu závodu (a zastaveniu danej simulácie) a presunu do zobrazenia tabuľky výsledkov. Server odošle klientom informácie o celkovom poradí klientov, ktorý je však závislý na type pretekov (uložené v premennej triedy *SServer::gameStyle*, určený administrátorom pomocou správy GAMESTYLE). Tabuľka 3. popisuje spôsob vyhodnocovania jednotlivých závodov a udalosti nastávajúce po prijatí správy STARTGAME od administrátora.

Tabuľka 3:

Štýl hry / vlastnosť	Poradie po závode	Akcia po STARTGAME
0 - single race	Určené poradím dojazdu vozidiel do finishu posledného kola (cieľa).	ukončenie severa a odpojenie klientov
1 - tournament	Vozidlám sú podľa poradia v cieli priradené body, ktoré sa po každom závode sčítajú. Poradie po každom preteku sa určuje podľa počtu bodov.	pokračuje sa ďalším závodom
2 - knock out	Poradie je určené rovnako jako pri single race.	posledný hráč je vyradený z hry (mení sa na spectatora), nasleduje ďalší závod

Týmto boli popísané všetky podstatné aspekty fungovania serverovej časti AnoRaSi. Ďalšie informácie je možné nájsť napr. v programátorskej dokumentácii a v kapitole 5 - o sieťovej komunikácii.

Kapitola 4 – Klient / zobrazenie simulácie

Úlohou klientskej časti aplikácie je umožňovať ľuďom – hráčom – pripájanie k serveru a zobrazovanie a ovládanie simulácie na ich počítači. Je pritom rozdelená na dva celky líšiace sa účelom a aj spôsobom použitia – menu a vlastné zobrazovanie simulácie. Nasledujúce kapitoly v skratke popisujú prácu s knižnicami použitými v klientovi a spôsob ich integrácie za cieľom vytvorenia grafického a zvukového prostredia.

4.1 Knižnice Irrlicht a OpenAL++

Knižnica Irrlicht je opensource projekt snažiaci sa o vytvorenie výkonného, objektovo orientovaného, multiplatformového 3D grafického engine-u. O jeho kvalite svedčí aj jeho používanie mnohými (aj komerčnými) projektami.

Prvým krokom pri používaní knižnice je vytvorenie jej základného objektu – *IrrlichtDevice*. Ten je možné vytvoriť aj volaním funkcie createDevice, ktorej parametrami sú napríklad rozmery grafického rozhrania (v prípade celoobrazovkového režimu rozlíšenie obrazovky) a použitý grafický driver. Tých Irrlicht poskytuje niekoľko, pričom pre rôzne platformy je zoznam použiteľných driverov rôzny. Napr. v systéme Linux sú to dva softvérové drivery a OpenGL, Windows verzia podporuje okrem už spomenutých aj DirectX 8 a DirectX 9 režimy. Používané grafické rozhranie ani jeho rozlíšenie nie je možné za behu jednoducho meniť – v prípade potreby zmeny je teda najvhodnejšie reštartovať celú aplikáciu.

Knižnica Irrlicht umožňuje vcelku jednoduchým a priamočiarym spôsobom vytvoriť udalosťami riadené grafické rozhranie, pričom sa sama stará o odstránenie väčšiny nepotrebných objektov (napr. dealokácia pamäti zabranej dátovými štruktúrami tlačidla po jeho odstránení). Komunikácia aplikácie s knižnicou prebieha priamo pomocou volaní jej funkcií a metód jej tried. Opačným smerom knižnica využíva systém callback funkcií, zapuzdrených do objektov. Princíp fungovania knižnice bude viac zrejmý z popisu použitia v AnoRaSi, preto upustíme od ďalších obecných informácií o nej. Okrem toho je možné kompletnú dokumentáciu knižnice aj s príkladmi použitia nájsť na webe [4].

Okrem grafického výstupu je v simulátore automobilových závodov samozrejme žiadúca aj zvuková komunikácia aplikácie s hráčom. Pôvodná špecifikácia bakalárskej

práce určovala na tento účel použitie knižnice OpenAL. Pretože je však táto knižnica príliš obecná a nízkoúrovňová, bolo rozhodnuté o použití jej objektovej nadstavby – OpenAL++. Táto nadstavba umožňuje jednoduchšie vytváranie zvukového prostredia a jej použitím je výsledný kód aplikácie značne prehľadnejší. Implementovaná je však pomocou volaní funkcií OpenAL. Okrem toho pracuje vo viacerých vláknach, takže plynulosť zvukového výstupu je do značnej miery nezávislá na vyťaženosti zvyšnej aplikácie. Vďaka použitiu vlákien ale OpenAL++ vyžaduje ku svojmu fungovaniu aj knižnicu OpenThreads. Tá je taktiež objektovo orientovaná a kvôli zachovaniu homogénnosti prostriedkov použitých na vývoj aplikácie je táto knižnica použitá na vytváranie všetkých vlákien v klientskej aj serverovej časti aplikácie AnoRaSi. Explicitná inicializácia OpenAL++ nie je potrebná, na rozdiel od OpenThreads. Spolu s grafickým prostredím je teda inicializovaná aj OpenThreads.

4.2 Grafické menu aplikácie

Potom, ako sú vytvorený hlavný objekt knižnice Irrlicht – device, môže začať aplikácia vytvárať grafické prostredie. Prvou časťou aplikáciou, s ktorou príde užívateľ do styku ja v AnoRaSi menu – pomocou neho je možné ovládať celú zvyšnú časť aplikácie (až na samotné ovládanie vozidla v simulácii, viď ďalej).

Objektový model knižnice Irrlicht síce neprikazuje striktne využívať v kóde objektovú štruktúru, jej použitie však výrazne sprehľadňuje kód. Preto sú všetky dáta a funkcie hlavného menu umiestnené v jedinej triede *CMainMenu*. Tá je odvodená od triedy IEventReceiver, čo umožňuje prijímať správy od Irrlicht (pomocou metódy OnEvent).

Vstupným bodom triedy *CMainMenu* je metóda *run()*. Tá po svojom spustení volá metódu *GainControl()*, ktorá inicializuje niektoré vnútorné premenné triedy a nastaví vlastný objekt ako prijímateľa správ pre aktuálne okno Irrlicht. Ďalším krokom je vytvorenie samotného grafického prostredia menu. Kvôli jednoduchosti a prehľadnosti je menu rozdelené na niekoľko rôznych „scén“, ktoré sa po určitých akciách menia.

Výmenu jednotlivých scén zabezpečuje jediná metóda – *SwitchScreen()*, ktorej parametrom je scéna ktorá sa má zobraziť. Táto metóda si zároveň pamätá poslednú zobrazenú scénu a v prípade potreby dokáže odstrániť z obrazovky v novej scéne nepotrebné prvky. Celé menu je vytvorené pomocou GUI subsystému knižnice Irrlicht.

Ten umožňuje vytváranie jednoduchých objektov ako okná, tlačidlá, textové polia a zoznamy (list boxy). Každému z týchto objektov sa dá priradiť rodič a ID číslo. Je tak možné vytvárať napríklad tlačidlá vo vnútri okien, zadaním tohto okna ako rodiča tlačidla. ID číslo sa využíva neskôr, v prípade potreby odkázať sa na niektorý objekt GUI. Každá scéna má v enumerácii *_currentScreen* pridelené číslo. Okrem jednoznačnej identifikácie scén toto číslo slúži aj ako offset pre všetky ID čísla objektov vytvorených v rámci nich. Práve vďaka takto vytváraným ID je možné jednoducho identifikovať všetky prvky danej scény (a v prípadne potreby ich vymazať). Vytváranie GUI objektov v metóde *SwitchScreen* je implementované jednoduchým volaním príslušných metód objektu IGUIEnvironment – čitateľ si to môže nájsť priamo v priloženom zdrojovom kóde aplikácie. Špeciálnymi druhmi scén sú CONNECT a GAMEPLAY. Po vykreslení scény CONNECT totiž dôjde k pokusu o pripojenie sa k serveru pomocou užívateľom zadaných parametrov (adresa, port a typ pripojenia - hráč/pozorovateľ). V prípade, že je pripojenie úspešné, metóda znovu rekurzívne volá sama seba a nastavuje scénu PREGAME. V opačnom prípade je zobrazená scéna CANTCONNECT, zobrazujúca chybové hlásenie. Scéna GAMEPLAY slúži na prepínanie klientskej časti aplikácie na zobrazenie simulácie, popis celého postupu je uvedený ďalej v tejto sekcii.

Po vytvorení úvodného grafického rozhrania sa dostáva metóda *run()* do slučky, ktorá zabezpečuje v každom kroku nové prekreslenie obrazovky a spracovanie správ od grafickej knižnice a zo sieťového pripojenia k serveru (ak je nadviazané). Táto slučka prebieha dovtedy, kým užívateľ nezatvorí okno aplikácie (napr. vo Windows „krížikom“ v pravom hornom rohu), respektívne kým nedá príkaz k ukončeniu priamo cez grafické menu.

Spracovanie správ z knižnice sa vykonáva metódou *OnEvent()* (ktorá je volaná priamo knižnicou Irrlicht). Podľa aktuálnej scény je vybraná množina udalostí, na ktoré bude menu reagovať, všetky ostatné správy sú „zahodené“. Čo sa menu týka, obsluhuje vždy len dva druhy gui správ: stlačenie tlačidla v GUI a stlačenie tlačidla na klávesnici. Druhý menovaný druh vo väčšine scén menu nahradzuje použitie klávesových skratiek – a pomocou rôznych skokov vo vnútri metódy vyvoláva udalosti rovnaké ako má stlačenie príslušného gui tlačidla. Výnimkou sú scény (v súčasnej implementácii to je jediná – scéna PREGAME), kde sa vyžaduje textový vstup. V nich stlačenie tlačidla Enter znamená ukončenie zadávania textu a povolenie k ďalšiemu spracovaniu (ostatné

tlačidlá klávesnice sa používajú na zadávanie textu). Metóda *OnEvent* vykonáva na základe vstupu od užívateľa viacero činností. V prípade potreby volá metódu *SwitchScreen()* na zmenu aktuálnej scény, pričom napríklad v scéne SETTINGS ešte pred zmenou scény ukladá do vnútorných štruktúr užívateľom zadané nastavenia.

Špeciálne postavenie majú scény CREATE a JOIN. Ak sa užívateľ pokúsi uzavrieť prvú zmieňovanú scénu tlačidlom OK, dôjde na vybranom porte k spusteniu serverovej – pomocou volania metódy *StartServer()*. Nasledujúci postup je rovnaký ako v prípade odsúhlasenia scény JOIN: zaznamenajú sa údaje, ku ktorému serveru a akým spôsobom sa chce klient pripojiť (ako hráč, alebo ako pozorovateľ) a dôjde k prepnutiu na scénu CONNECT. Tá však neprijíma od užívateľov žiadny vstup a slúži iba na informovanie o stave pripájania.

Scéna PREGAME je zobrazená hneď po pripojení k serveru. Jej účelom je poskytovať užívateľom možnosť vybrať si druh vozidla, nick a umožniť im komunikovať medzi sebou pomocou jednoduchého chatu. Klávesa enter má v tejto scéne špeciálne vlastnosti. V prípade, že má jedno z jej textových polí nastavený focus, po stlačení enteru dôjde k odoslaniu daného údaju serveru cez socket. Odoslanie nicku sa vykonáva pomocou správy NICK, text chatu pomocou správy CHAT. V prípade stlačenia jedného z tlačidiel na výber vozidla alebo mapy (to je možné len v prípade, že je užívateľ administrátorom) dôjde k prepnutiu na jednu zo scén SELECT_CAR a SELECT_MAP – v praxi to znamená zobrazenie okna. Tieto okná=scény sú si funkčnosťou veľmi podobné a ich jediný účel je umožnenie užívateľovi vybrať jednu z ponúkaných položiek. Po uzavretí týchto okien tlačidlom OK dôjde k odoslaniu textu vybratej položky na server (správami CAR, resp. MAP). Okrem toho je možné v scéne PREGAME stlačiť tlačidlo „Game style“. Tým dôjde k zobrazeniu okna s možnosťou výberu počtu kôl na závod a typu pretekov (single race, knockout, tournament). Po odsúhlasení okna (znovu tlačidlom OK) sa nové nastavenia uložia do vnútorných štruktúr. Začiatok pretekov môže ako jediný iniciovať administrátor – stlačením tlačidla „Play“. Tým dôjde najprv k odoslaniu správy GAMESTYLE s nastavením pretekov a následne aj k započatiu hry správou STARTGAME.

Po ukončení jedného závodu je vždy zobrazená scéna RESULTS, v ktorej je zobrazené poradie jazdcov v rámci celých pretekov. Okrem iného táto scéna umožňuje prerušenie spojenia so serverom (podobne ako PREGAME). V prípade, že tak spraví administrátor, dôjde následne aj k ukončeniu servera. V oboch prípadoch je na tento

účel volaná metóda *ServerStop*. Administrátor má aj právo stlačením tlačidla „Continue“ oznámiť serveru, že preteky môžu pokračovať ďalším závodom (v tomto prípade je na server odoslaná správa STARTGAME).

Druhým typom spracovávaných správ sú správy prijaté cez sieťový socket od servera. Tento druh správ sa spracováva v metóde *ProcessSocketMessages*. Hlavnou súčasťou tejto metódy je slučka postupne čítajúca a spracovávajúca z fronty príchodzie správy od servera. V prípade prerušenia spojenia so serverom sa stará o zmenu obrazovky na CONNECTION_LOST. Tabuľka 4. poskytuje stručný prehľad jednotlivých správ spolu s krátkym popisom ich významu.

Tabuľka 4: Správy prijímané od servera

Správa	Popis	Akcia
ADMIN	Oznámenie servera klientovi, že nadobudol administrátorské práva.	Nastavenie premennej triedy adminMode, povolenie administrátorských úkonov v GUI.
STARTGAME	Server začal prípravu na simuláciu	Klient sa prepne do módu zobrazovania simulácie
CHAT	Chatová správa	Zobrazenie správy v GUI
USERS	Zoznam pripojených klientov a botov	Zobrazenie zoznamu v GUI
LIST_CARS LIST_MAPS	Zoznam dostupných vozidiel / máp	Uloženie zoznamu do vnutorných štruktúr, kvôli prípadnému neskoršiemu zobrazeniu...
CAR	Zmena typu klientovho vozidla	Zobrazenie typu vozidla v GUI
MAP	Zmena aktálnej mapy	Zobrazenie názvu mapy v GUI
NICK	Zmena nicku klienta	Zobrazenie nového nicku v GUI
JOIN	Informuje klienta, že bol serverom úspešne zaznamenaný jako nový hráč	Povolenie niektorých prvkov v GUI scény PREGAME – najmä tlačidlo na zmenu vozidla
SPECTATE	Informuje klienta, že ho server od daného okamihu „vníma“ jako pozorovateľa	Zakázanie niektorých prvkov v GUI scény PREGAME – najmä tlačidlo na zmenu vozidla

Prepnutie do módu zobrazovania simulácie sa deje nastavením scény na GAMEPLAY. *SwitchScreen()* v tomto prípade najprv odstráni všetky ním vytvorené grafické prvky a následne vytvorí objekt typu *CGame* a volá jeho metódu *run()*, ktorá ďalej preberie riadenie aplikácie. Po návrate z nej podľa návratovej hodnoty môže buď

dôjsť k ukončeniu programu (v prípade že bolo zatvorené okno simulátora), alebo sa zavolá metóda *GainControl()* a zobrazí scéna RESULTS (pri ukončení závodu serverom – po prejazde všetkých závodníkov cieľom) alebo MENU (Ak o ukončenie žiadal klient). V druhom prípade dôjde taktiež k ukončeniu spojenia so serverom, prípadne aj k ukončeniu servera (ak je nejaký spustený).

4.3 Zobrazovanie simulácie

Po prijatí správy STARTGAME klientská časť AnoRaSi vytvorí triedu *CGame* a zavolá jej metódu *run()*, čím táto trieda preberá riadenie zobrazovania a sieťovej komunikácie. Podobne ako u menu, aj táto metóda na začiatku inicializuje vnútorné premenné a nastaví vlastný objekt ako príjemcu správ od knižnice Irrlicht. Ďalšia a posledná podobnosť s menu je v rozdelení tejto časti aplikácie na niekoľko scén – v tomto prípade je však pevne určené ich poradie. Na zmenu scény je tu použitá iná metóda – *switchToNextScene()*. Pritom ako prvá scéna je zobrazená (pomocou metódy *doScene0_introScreen*) tzv. „Loading screen“, ktorá je zobrazená po celú dobu zahajovania simulácie (tzn. načítania dát na strane servera a následne klienta). V ďalšom kroku dôjde opäť k zmene scény (volaním *doScene1_serverWait*), tentokrát však nie je nič vykreslené. Táto scéna vykonáva nekonečný cyklus, kým od servera nepríde správa GETREADY (resp. kým nebude prerušené spojenie so serverom, v takom prípade dôjde aj k vyskočeniu z metódy *run()*). Po jej obdržaní dôjde k ďalšej zmene obrazovky a k zavolaniu metódy *doScene2_loadData()*.

Táto scéna je prvá, v ktorej klient vykonáva podstatnú činnosť. Na začiatku scény prijíma od servera správy definujúce vzhľad simulácie, pričom ukončenie tohto bloku je „označený“ správou WAITING. Správy spracovávané klientom sú vypísané tabuľke 5. Neuvedené správy ostávajú klientom nespracované.

Tabuľka 5: Konfiguračné správy pre klienta

Správa	Popis
WORLD	Oznamuje klientovi názov mapy použitej na strane servera.
CAR	Definuje vozidlo jedného z hráčov (definície prichádzajú v poradí vzostupne podľa ID)
YOURID	Oznamuje klientovi (ak sa jedná o hráča, nie o pozorovateľa) číslo jeho vozidla (vďaka tomu je zrejmé, na ktoré vozidlo sa má neskôr zamerať kamera)
PLAYER	Štandardná správa určujúca pozíciu vozidla a jeho súčastí. Je zaručené, že táto správa príde až potom, ako sú prijaté všetky správy CAR.

Po načítaní správy CAR dôjde priamo k vytvoreniu objektu typu *CCarModel* a volaním jeho metódy *Load()* dôjde k inicializácii dátových štruktúr potrebných pre zobrazenie tohto vozidla (a dodatočne je odkaz na objekt uložený do zoznamu vozidiel vo vnútri triedy). Správa WORLD nemá počas čítania správ žiadny účinok, názov mapy sa iba uloží do lokálnej premennej. Po ukončení konfiguračného bloku je hodnota tejto premennej využitá a dôjde k načítaniu konfiguračného súboru danej mapy. Následne dôjde volaním metódy *doScene2_loadData_loadMap()* k načítaniu všetkých objektov v ňom definovaných (skybox – statické prostredie tvorené hranami kocky, obklopujúce celú mapu; všetky objekty spolu s textúrami). Ďalej je vytvorená trieda *CCandies*, starajúca sa okrem iného o zobrazovanie checkpointov a finishu (takže zároveň sú tieto objekty umiestnené do grafickej scény). Dôjde aj k načítaniu textúr použitých na zobrazenie užívateľského rozhrania aplikácie – tachometra, otáčkomera a pod. Na záver sú pomocou vytvárania objektov Source knižnice OpenAL++ do pamäti načítané zvuky motorov jednotlivých vozidiel. Potom, ako je celé grafické a zvukové prostredie inicializované, dôjde k zaslaniu správy READY na server. Ňou klient deklaruje svoju pripravenosť – to, že na jeho strane už boli načítané všetky dáta a že sa prepína do scény zobrazujúcej samotnú simuláciu.

Metódou, ktorá sa stará o zobrazenie simulácie, interakciu užívateľa s ňou a o komunikáciu so serverom, je *doScene3_main()*. Ako už viacero metód tohto programu, aj táto je volaná v cykle a je možné pomyslene ju rozdeliť na niekoľko krokov, z ktorých každý ma iný účel:

- Na začiatku je volaná metóda *processControls(),* ktorá slúži najmä na spracovanie vstupu užívateľa týkajúceho ovládania vozidla. Vstup samozrejme nie je možné použiť priamo – napríklad v prípade plynového pedálu by to znamenalo, že by bolo hráčovi umožnené len jeho úplné stlačenie alebo úplné uvoľnenie. Ďalším príkladom by mohlo byť otáčanie kolies, ktoré by mohlo nadobúdať len stavov plné otočenie doľava, priamy smer a plné otočenie doprava. Takéto riešenie by samozrejme bolo veľmi vzdialené realite a navyše by vozidlo nebolo dobre ovládateľné. Preto je jemné nastavovanie jednotlivých hodnôt vyriešené postupnou konvergenciou k hodnotám požadovaným používateľom. To sa samozrejme netýka prevodového stupňa, ktorého hodnota sa zväčšuje/zmenšuje priamo.
- Skontroluje sa stav spojenia so serverom – ak došlo k jeho prerušeniu, zobrazovanie simulácie je ukončené a riadenie programu je riadené do menu.
- V prípade že užívateľ nie je pozorovateľom, dochádza k odoslaniu správy CONTROL (popisujúcej stav ovládania jeho vozidla) na server.
- Ďalej sú v určitých intervaloch na server posielané správy PING (s aktuálnym časom ako parametrom), pomocou ktorých klient zisťuje rýchlosť jeho odozvy.
- Konečne dochádza k spracovaniu správ zo servera. Najčastejšie prichádzajúcimi správami sú PLAYER, SIMSTATUS a OBJECTS, pričom o spracovanie týchto správ sa starajú postupne správy *processMessage_PLAYER*, *processMessage_SIMSTATUS* a *processMessage_OBJECTS*. Správa PLAYER popisuje polohu a stav ovládania vozidla v simulácii. SIMSTATUS obsahuje aktuálny čas simulácie a zároveň informácie o prechodoch klientov checkpointom alebo finishom. Konečne správa OBJECTS popisuje polohy objektov terénu. Ďalšou možnou prijatou správou je PONG, ktorá je odpoveďou na klientom odoslanú správu PING. Dĺžka odozvy sa potom počíta ako rozdiel aktuálneho času a čísla v parametri správy PONG (keďže server odpovedá na každú správu PING zaslaním správy PONG s rovnakým parametrom). V prípade iniciácie reštartu závodu administrátorom sú všetkým klientom odoslané správy RESTART, po ktorých majú vynulovať vnútorné čítače počtu kôl a pod. Po tejto správe dôjde k novému načítaniu simulácie na strane servera. Keď je dokončené, klient obdrží správy PLAYER, SIMSTATUS a OBJECTS s novými vlastnosťami simulácie a nakoniec správu WAITING, ktorá oznamuje koniec reštartu servera. Po detekcii konca závodu (na strane servera) je klientovi

odoslaná správa FINISH, ktorej parametrami je výsledné poradie hráčov. Toto poradie je (pre neskoršie zobrazenie v menu) uložené do vnútornej premennej triedy (*CGame*) a nakoniec dochádza k opusteniu tela metódy *run()*.

- Po spracovaní prípadných správ dochádza nastaveniu interpolovaných pozícií vozidiel (čomu sa venuje ďalšia sekcia) a k nastaveniu pozícií zvukových zdrojov v simulácii.
- Ďalej je nastavená pozícia kamery a jej cieľ (resp. smer otočenia). Tieto dve polohy však nie sú nastavované priamo, ale ako istý druh priemeru nadobudnutých polôh. Napríklad pre polohu cieľa kamery to znamená, že v istom poli je udržiavaný zoznam polôh, do ktorých sa vozidlo dostalo, spolu s časom a faktorom priamo úmerným rýchlosti pohybu vozidla v danom okamihu. Tento zoznam je pri každom získavaní „priemernej pozície" pretriedený tak, aby obsahoval len pozície z určitého časového obdobia. Výsledná poloha je potom určená ako vážený priemer pozícií. V tomto kroku je zároveň nastavená aj poloha objektu Listener knižnice OpenAL++. Tým dochádza k synchronizácii polohy kamery a virtuálneho mikrofónu v scéne (a tým aj zrakového a zvukového vnemu užívateľa).
- V prípade že uplynie 5s od štartu, sú zo scény „vymazané" objekty znázorňujúce štartovacie posty.
- Ďalší krok je výpočtovo najzložitejší a vykonáva ho knižnica Irrlicht – ide o zobrazenie vlastnej scény na grafické zariadenie.
- Potom nasleduje druhá fáza zobrazovania – tentokrát sú zobrazené grafické prvky aplikácie – otáčkomer, tachometer, aktuálny prevodový stupeň, všetky prvky subsystému gui a prípadne doplňujúce debug údaje. Na záver je do statusbaru scény vypísaný text informujúci klienta o aktuálnom FPS (počet prekreslení obrazovky za sekundu), ping dobe a o aktuálnom čase simulácie.

Podobne ako trieda *CMainMenu*, aj *CGame* obsahuje metódu *OnEvent*, ktorá umožňuje spracovávanie udalostí prijatých od užívateľa. V prípade *CGame* nás znovu zaujímajú len stlačenia klávesov a prípadne tlačidiel gui. Priamo v tejto funkcii sú spracované stlačenia klávesov používaných na otáčanie kamery okolo vozidla, zmenu cieľa kamery a tlačidlo escape. V prípade stlačenia resp. uvoľnenia iných tlačidiel je informácia o ich stave uložená do poľa *kbd_states* – ktoré je potom využívané už spomínanou funkciou *processControls()*.

4.4 Aproximácia polôh telies v scéne

Vzhľadom na možné nepravidelnosti v doručovaní správ cez sieť je potrebné použiť nejaký spôsob vyrovnávania týchto nepravidelností. Každý pohyblivý objekt je na klientskej strane aplikácie reprezentovaný triedou *CModelAttr*. Tá vo vnútri obsahuje ukazateľ na objekt Irrlicht, ktorého pozíciu ovláda. Plynulosť pohybu telies je dosiahnutá pomocou bufferovania pozícií. Od servera časom chodia informácie o pozíciách jednotlivých objektov spolu s časovými údajmi, kedy sa v daných miestach nachádzali. Pozícia je spolu s časom umiestnená do vlastného zoznamu každého z týchto objektov (pomocou metódy triedy *CModelAttr::setPosRot()*). Pred každým vykresľovaním scény Irrlicht-om je volaná metóda *CModelAttr::move()*. Tá nastaví aproximovanú pozíciu telesa v časovom okamihu danom jej parametrom.

Základný princíp jej fungovania je nasledujúci: Najprv sú zo zoznamu pozícií vymazané všetky pozície s časom menším ako je čas daný parametrom. Potom sa zo zoznamu „vezmú" prvé dve položky a lineárnou aproximáciou z ich hodnôt je vypočítaná výsledná pozícia a rotácia telesa.

Aby tento postup fungoval dobre, je potrebné zaviesť do nastavovania pozícií objektov istú umelú latenciu. V opačnom prípade by mohlo dôjsť k neustálemu úplnému vyprázdňovaniu bufferu pozícií, čo by malo zlý vplyv na plynulosť pohybu telies. Táto latencia je znovu emprická hodnota (konkrétne 80 ms). Je nastavená tak, aby si hráč nevšimol časový posun napríklad medzi stlačením tlačidla otáčania kolesom a zobrazením tohto otočenia – ale na druhej strane aby nedochádzalo (v rámci možností) ani k spomínanému vyprázdňovaniu buffera na pomalších linkách.

4.5 Herné prvky scény

Podobne ako v serverovej časti, aj v klientskej je potrebné si pre jednotlivé vozidlá udržiavať zoznam prejdených checkpointov a finishov. Rozdiel je ale v tom, že kým serverovej časti stačí poznať poradie dojazdu vozidiel do cieľa, v klientskej aj poradie hráčov počas závodu[13]. Tieto zoznamy sú udržiavané priamo v štruktúrach popisujúcich ostatné vlastnosti vozidiel – *CModelAttr*. Ako už bolo naznačené, server upozorňuje na prejdenie vozidla jedným z týchto objektov správou SIMSTATUS, ktorá sa spracováva

[13]Táto potreba nie je samoúčelná. Dôvodom je potreba priebežného zobrazovania poradia a vzájomnej „vzdialenosti" hráčov.

v metóde *procesMessage_SIMSTATUS()*. Tá však túto správu (v už „dekódovanej" podobe) ďalej posúva triede *CCandies*, konkrétne jej metódam *Checkpoint_Crossed* a *Finish_Crossed.*

Obe využívajú tieto správy na postupné zobrazovanie a skrývanie neprejdených objektov v závislosti na už popisovanej hodnote *Checkpoints/order.* Pri sekvenčnom poradí je zobrazovaný vždy len jeden objekt z množiny checkpointov a finishu - ten, ktorým musí vybrané vozidlo[14] prejsť najbližšie aby bol „uznaný". V prípade náhodného poradia sú na začiatku kola zobrazené všetky checkpointy a finish je skrytý. Postupným prechádzaním checkpointov dochádza k ich skrývaniu a keď sú prejdené všetky, dôjde k zobrazeniu finishu.

V prípade, že hráč zmení vozidlo sledované kamerou, musí dôjsť aj k úprave viditeľnosti jednotlivých objektov tak, aby zodpovedali stavu tohto vozidla – čo sa týka prejazdu jednotlivými objektami. To sa po zmene cieľu kamery dosiahne volaním metódy *Ccandies::RenewAll()*.

Okrem toho majú spomínané správy aj iné využitie – určenie poradia hráčov v závode. To sa znovu prepočítava a vypisuje do prvkov grafického rozhrania volaním metódy *CCandies::PlayersResort()*. V prípade náhodného poradia prechádzania checkpointami je poradie hráča v pretekoch dané vzťahom

$$PočetPrejdenýchKôl * (PočetCheckpointovTrate + 1) + PočetPrejdenýchCheckpointov$$

pričom väčšie číslo značí lepšiu pozíciu. Pri sekvenčnom poradí je poradie určené zrejmým spôsobom. Už zložitejšie je zistiť časový odstup jednotlivých dvojíc vozidiel. V prvom prípade nie je možné určiť smerodajný časový odstup, preto je vždy zobrazovaný len rozdiel počtu prejdených kôl a checkpointov. Druhý prípad už túto možnosť poskytuje – stačí odčítať čas prejdenia daného checkpointu druhým hráčom v poradí od času prejdenia toho istého checkpointu prvým hráčom.

Tým je v stručnosti popísané fungovanie celej klientskej časti aplikácie. Bolo by možné detailnejšie popísať postup načítania jednotlivých grafických objektov a ich umiestnenia do scény. No podobne ako pri serverovej časti, aj tu platí – spraviť si detailnejší obraz o fungovaní týchto častí aplikácie je možné náhľadom do zdrojových kódov, prípadne programátorskej dokumentácie.

[14]Teda to vozidlo, na ktorého sledovanie je práve nastavená kamera.

Kapitola 5 – Sieťová komunikácia

Obe časti aplikácie používajú na komunikáciu medzi sebou tú rovnakú triedu – *AsyncComm*. Táto trieda umožňuje komunikovať pomocou socketov asynchrónne. To v praxi znamená nulové zdržanie v situáciách, keď na vstupe socketu nie sú dostupné žiadne dáta. Parametrom konštruktora sú je pointer na už vytvorený objekt typu *Socket*, nad ktorým trieda *AsyncComm* prevezme kontrolu (čo napríklad znamená že odstraňovaní objektu *AsyncComm* je jeho deštruktorom odstránený aj ním ovládaný *Socket*). Na zistenie aktívnosti spojenia je možné použiť metódu *Alive()* a na ukončenie spojenia explicitne metódu *disconnect()*. Metódami určenými na komunikáciu sú *SendString()* (slúži na odosielanie správ „druhej strane socketu“) a *ReadString()* (po jednej vracia prijaté správy; v prípade že žiadna nečaká na vyzdvihnutie, vráti chybovú hodnotu).

Tieto funkcie dokážu teda odoslať a prijať správu akéhokoľvek formátu. Vnútorne dochádza pred odoslaním ku konverzii niektorých znakov tak, aby mohli byť prijímané správy od seba správne oddelené (a následne sú znovu konvertované do pôvodného tvaru). Kvôli prehľadnosti bol však pre komunikáciu klientov a servera stanovený jednotný spôsob vytvárania týchto správ.

Každá z nich má tvar:

```
%:NAZOV_SPRAVY
ID1: hodnota 1
ID2: hodnota 2
...
```

Jednotlivé riadky sú od seba oddelené znakom konca riadku (kód 0x0a), pričom povinný je len prvý riadok určujúci typ správy. Každý z riadkov má pritom tvar „Názov:Hodnota“. V prípade, že sa pomocou správy prenáša jediná hodnota (ako je to napríklad v prípade správy NICK), používa sa konvencia zvoliť názov položky pre túto hodnotu DATA. Popis položiek jednotlivých druhov správ je možné nájsť v programátorskej príručke.

Aplikácia AnoRaSi ako taká, ani triedy *AsyncComm* a *Socket*, neriešia chyby v komunikácii. Spoliehajú sa totiž na vcelku účinnú detekciu a opravu chýb obsiahnutú priamo v protokole TCP/IP. V prípade nedostatočnej priepustnosti alebo kvalite spojenia však môže dochádzať k oneskoreniu doručenia paketov stroju na opačnej strane. Tento

problém je riešený len na strane servera a len pri odosielaní správ popisujúcich pozície objektov v simulácii.

V prípade, že oneskorenie medzi požiadavkom na odoslanie správy a skutočnou dobou, kedy je odosielaný cez socket je väčší ako 200 ms (to už je hodnota ohrozujúca hrateľnosť), je daná správa zahodená a pokračuje sa ďalšou správou (resp. prvou najbližšou správou s latenciou menšou ako 200 ms). Tým síce môže dôjsť k trhanému pohybu objektov simulácie, na druhej strane je to však prijateľnejšie riešenie, ako keby bolo zobrazenie simulácie oneskorené (pričom pri konštantne nízkej šírke pásma by sa oneskorenie časom zhoršovalo – čím ďalej viac správ by totiž čakalo na odoslanie).

Dôvod, prečo sa podobný spôsob „zahadzovania" oneskorených odchodzích správ nepoužíva aj v ostatných častiach aplikácie je vcelku prostý. Vynechanie jednej či niekoľkých správ o pozíciách objektov neohrozuje simuláciu - iba skresľuje vzhľad scény na strane klienta. Naopak, napríklad vynechanie doručenia správy o začiatku hry klientovi by mohlo znamenať nesprávnu inicializáciu celého systému klienti-server.

Kapitola 6 – Záver

Zadaním bakalárskej práce bolo vytvorenie aplikácie, ktorá simuluje virtuálne prostredie. Okrem toho mala umožňovať vzájomné súťaženie jednotlivých hráčov. V zadaní práce boli špecifikované aj knižnice, ktoré mali byť použité k vytvoreniu výslednej aplikácie. Keďže však bola práca pokračovaním pred tým vyvíjaného ročníkového projektu, bolo potrebné držať sa zároveň aj pôvodnej špecifikácie programu.

Ako bolo uvedené v sekcii 4.1, odklon od zadania bakalárskej práce nastal pri výbere knižnice určenej na zvukový výstup. K tejto zmene došlo v záujme kvalitnejšie, prehľadnejšie a robustnejšie naprogramovanej aplikácie. Napriek tomu je knižnica OpenAL používaná naďalej – aj keď nepriamo.

Všetky ostatné podstatné požiadavky uvedené v špecifikácii boli splnené – s výnimkou niekoľkých, pre finálny vzhľad a funkčnosť aplikácie nedôležitých vlastností. Nastala však aj opačná situácia a aplikácia implementuje niektoré vlastnosti nad rámec špecifikácie. Príkladom je napríklad dôležitá možnosť použiť na vytvorenie prostredia pomocou modelov trimesh, ktorá značne rozširuje možnosti tvorby virtuálnych prostredí.

Počas vývoja bola aplikácia podrobená vcelku rozsiahlym testom na stabilitu a správne fungovanie. Všetky závažné problémy boli vyriešené a v súčasnom štádiu je v podstate možné prehlásiť aplikáciu za stabilnú a použiteľnú.

Prínosom práce je najmä poskytnutie referenčnej implementácie reálneho simulátora za použitia daných prostriedkov (knižníc). Spolu s programátorskou príručkou môže tvoriť solídny základ nových projektov. Po odovzdaní práce bude program vystavený spolu so zdrojovými kódmi na internet, odkiaľ si ho môžu všetci záujemcovia stiahnuť.

Pre autora mala práca prínos najmä v oboznámení sa so všetkými použitými knižnicami a algoritmami a s princípom ich nasadenia v reálnej aplikácii. Bol to aj prvý väčší projekt, ktorý vytváral podľa presnej špecifikácie (miesto voľného, postupného vývoja), čo môže byť do budúcnosti veľmi dôležitá skúsenosť.

Táto bakalárska práca sa venovala viac obecnému fungovaniu aplikácie a bola snaha vyhýbať sa popisu štýlom programátorskej dokumentácie. Tá je uložená, spolu s užívateľskou a inštalačnou príručkou, na priloženom médiu. Okrem toho sa na ňom nachádza vlastná aplikácia, jej dáta a zdrojové kódy. Pre archívne účely je na médiu priložený aj kompletný subversion strom, zaznamenávajúci chronologicky posledné mesiace vývoja aplikácie.

Literatúra

[1] Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel: Moderní počítačová grafika, Computer Press 2004, ISBN: 80-251-0454-0

[2] ODE: Open Dynamics Engine - http://www.ode.org/

[3] Car Physics for Games: http://home.wxs.nl/~monstrous/tutcar.html

[4] Irrlicht: Lightning fast 3d realtime engine - http://irrlicht.sourceforge.net/

Prílohy