Gymnázium Jozefa Lettricha v Martine
Jozefa Lettricha 2, 036 01 Martin

Stredoškolská odborná činnosť

č. odboru 11 - Informatika

Tvorba programu pre vizualizáciu a analýzu cestnej infraštruktúry

2020
Martin

Anton Kica
Septima B

Gymnázium Jozefa Lettricha v Martine
Jozefa Lettricha 2, 036 01 Martin

Stredoškolská odborná činnosť

č. odboru 11 - Informatika

Tvorba programu pre vizualizáciu a analýzu cestnej infraštruktúry

2020
Martin

Anton Kica
Septima B

Čestné vyhlásenie

Prehlasujem, že prácu s názvom „Tvorba programu pre vizualizáciu a analýzu cestnej infraštruktúry“, som vypracoval samostatne s použitím uvedenej literatúry a súčasne, že som túto prácu neprihlásil a neprezentoval v žiadnej inej súťaži vyhlásenej MŠVVaŠSR.

V Martine 13.2.2020 .....

Obsah

0. Úvod

Simulácie sú moderný spôsob, ako objavovať a skúmať svet z inej perspektívy. Ukazujú svet a ľubovoľné okolie z rôznych a niekedy pútavých perspektív, ktoré sú niekedy nepredstaviteľné, neprístupné alebo veľmi ťažko prístupne.

Dôvody, prečo sme si zvolili túto tému, nie sú závažné ani kritické pre spoločnosť. V našom výbere hrala veľkú rolu naša ľudská zvedavosť, potreba a prianie naprogramovať niečo, čo by bolo zaujímavé. Náš každodenný pohľad na cesty nás naplnil záujmom vidieť, ako vyzerá každodenná cestná premávka aj z iného uhla, z takého, ktorého by sme videli rôzne ďalšie aspekty, ako napríklad efektivita, rýchlosť či jedonoduchosť alebo komplikovanosť.

Rozhodli sme sa teda vytvoriť softvér na simulovanie dopravy od základov. V tomto softvéri sme chceli uvidieť dopravu z vtáčej perspektívy, nielen z pohľadu chodca.

Pomocou tejto simulácie by sme mohli dokonca simulovať reálnu premávku, rôzne s ňou experimentovať, hľadať a navrhovať rôzne riešenia danej situácie bez nutnosti riskovať v realnom svete.

Chceli by sme ešte poďakovať pani profesorke Jane Nemilej za pomoc pri tvorbe SOČ.

1. Problematika a prehľad literatúry

1.1 Počítačový softvér

Softvér alebo počítačový softvér je kolekcia dát alebo počítačových inštrukcií, ktoré opisujú počítaču, ako pracovať. V počítačovej vede a v softvérovom inžinierstve chápeme pod pojmom počítačový softvér všetky informácie spracované počítačovými systémami, programami a dátami. Softvér je písaný programovacím jazykom a skladá sa z počítačového programu, knižníc a dát (dokumentácia alebo digitálne médium). Hardvér a softvér počítača sú navzájom potrebné a nevyhnutné pre ich funkčnosť. Programovacie jazyky sa používajú na komunikáciu medzi softvérom a hardvérom.

1.2 Programovací jazyk

Programovací jazyk je formálny jazyk, ktorý zahŕňa sadu inštrukcií produkujúcich rôzne typy výstupov. Tieto jazyky sa používajú v počítačovom programovaní na implementáciu algoritmov. Väčšina programovacích jazykov pozostáva z inštrukcií pre počítač. Existuje tisíc rôznych programovacích jazykov. Mnohé z jazykov sú písane v imperatívnej forme (sekvencia operácií na vykonanie), zatiaľ čo ostatné používajú deklaratívnu formu (špecifikovaný je očakávaný výsledok, nie postup jeho dosiahnutia).

Každý jazyk má svoju špecifickú syntax - formu, a jej sémantiku - význam. Programovacie jazyky môžeme rozdeliť v závislosti od miery abstrakcie na:

- nízke programovacie jazyky,
- vyššie programovacie jazyky,

a v závislosti a od spôsobu prekladu a spustenia na:

- kompilované jazyky,
- interpretované jazyky.

1.2.1 Nízke programovacie jazyky

Nízke programovacie jazyky poskytujú malú alebo žiadnu abstrakciu od funkcie procesora v počítači, sú tesne spojené s hardvérom. Sú veľmi rýchle a vyžadujú málo pamäte, no sú vcelku málo intuitívne. Tieto jazyky nevyžadujú kompilátor ani interpreter, avšak modernejšie jazyky používajú assembler. Medzi nízke programovacie jazyky patria napr. assembly laguage, machine language (strojový kód).

1.2.2 Vyššie programovacie jazyky

Vyššie programovacie jazyky poskytujú veľkú abstrakciu od funkcie procesora v počítači, nachádzajú sa v nich elementy prirodzených jazykov. Sú o niečo málo

pomalšie a vyžadujú väčšie množstvo pamäte. Sú jednoduché, veľmi prehľadné a rýchlejšie na vývoj softvéru. Tieto jazyky potrebujú byť pred samotnou exekúciou skompilované kompilátorom. Medzi vyššie programovacie jazyky patria napr. Fortran, COBOL, C, C++, C#, Python.

1.2.3 Kompilované programovacie jazyky

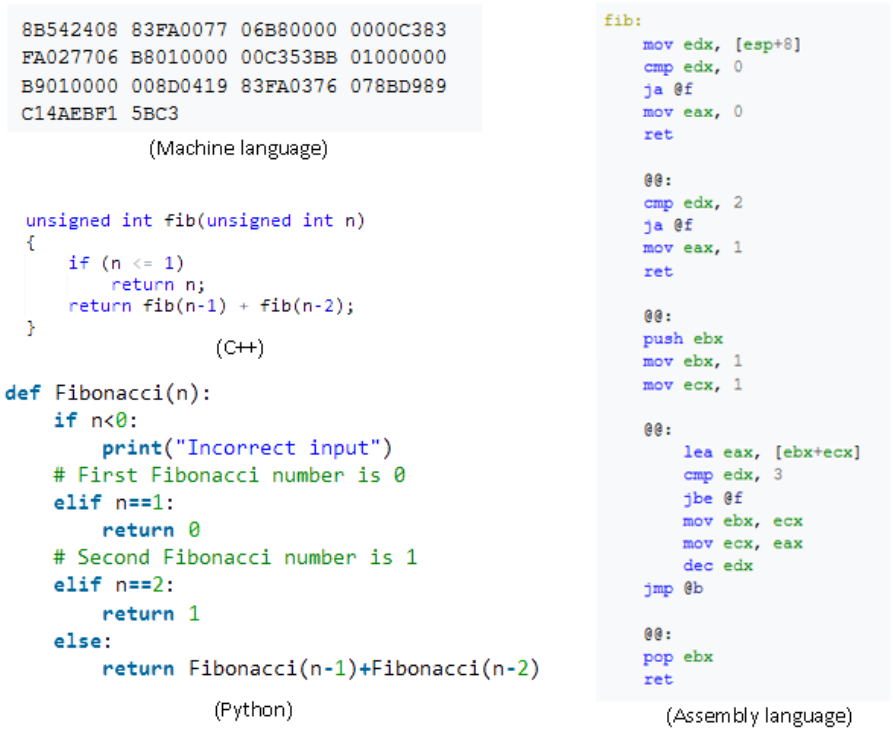
Pred exekúciou sú najprv preložené kompilátorom do nižšieho programovacieho jazyka. Kompilátor je schopný optimalizovať kód, čím zaistí vyššiu rýchlosť kódu.

1.2.4 Interpretované programovacie jazyky

Sú počas exekúcie preložené do medzikódu (medzijazyk) a následne vykonané, preto sú tieto jazyky pomalšie.

1.3 Algoritmus

„Algoritmus je konečná postupnosť presne definovaných inštrukcií na splnenie určitej úlohy. nazývame ho čiastočne správnym, ak v prípade, že skončí, dáva vždy správne výsledky a nazývame konečný, ak pre ľubovoľné vstupné údaje skončí v konečnom čase. Algoritmus, ktorý je čiastočne správny a konečný, sa nazýva správny. Program je algoritmus napísaný v programovacom jazyku.“ [1]



Obrázok 1: Algoritmus pre výpočet Fibonacciho postupnosti v rôznych jazykoch, Anton Kica

1.4 Vývoj softvéru

Vývoj softvéru je proces tvorby, špecifikovania, programovania, dokumentovania, testovania a opravovania chýb, zakomponovaný v tvorbe a údržbe aplikácií a rôznych softvérových komponentov. Tento proces je veľmi často štruktúrovaný a naplánovaný a predstavuje každú časť medzi prvotnou predstavou podoby požadovaného softvéru a jeho konečnou podobou. Týchto krokov je niekoľko a ich zložitosť sa môže pri rôznych požiadavkách prirodzene meniť (niekedy aj počet alebo ich poradie). Vo veľkej väčšine prípadov majú rovnakú fundamentálnu funkciu v procese tvorby a aj na seba postupne nadväzujú - výsledok minulého kroku ovplyvňuje nasledujúce.

Identifikovanie potreby

V prvom kroku sa vytvára nápad. Nápad môže byť smerovaný k vlastným potrebám – hobby, alebo k väčšiemu publiku – verejnosti. Námet na nápad je často ovplyvnený prieskumom trhu a jeho dopytom.

Plánovanie

V tejto fáze je známa predstava toho, čo sa bude vytvárať, ešte však nie je známe, ako presne. Zisťuje sa, čo všetko plány zahŕňajú, čo bude potrebné získať, jednoducho povedané, extrahujeme požiadavky.

Projektovanie

Keď už je všetko naplánované, prechádza sa do fázy projektovania. Už sú určené požiadavky a začína sa projektovať (navrhovať, tvarovať). Vytvára sa predbežný alebo detailný návrh hlavných prvkov s celkovým obrazom, ako vzájomne do seba zapadajú a sú od seba závislé. Programovací jazyk, prostredie, operačný systém a hardvér by mali byť už určené.

Implementácia, testovanie a dokumentácia

Všetky plány projektu sa programujú a kódujú. Nasledujúcim testovaním sa hľadajú chyby v realizácii, ktoré môžu byť spôsobené nedokonalým plánovaním alebo ľudským faktorom. Všetky nedostatky sú opravené. Tiež sa zisťuje výkon a reálne požiadavky programu. Následne je zdokumentovaný vnútorný dizajn pre budúcu údržbu alebo obohatenie ďalším vývojom (v niektorych prípadoch to znamená vytvoriť aj API).

Nasadenie a údržba

Finálny krok je nasadenie, s čím je spätá aj samotná údržba softvéru. Nasadenie nastáva po vhodnom testovaní a schválení produktu. Následne je produkt publikovaný a distribuovaný. Tiež sa zverejnia návody na jeho použitie a je zriadená podpora pre používateľov.

Údržbou a ďalším vývojom softvéru je zabezpečená oprava novoobjavených chýb alebo sa pridávajú nové možnosti požadované užívateľmi.

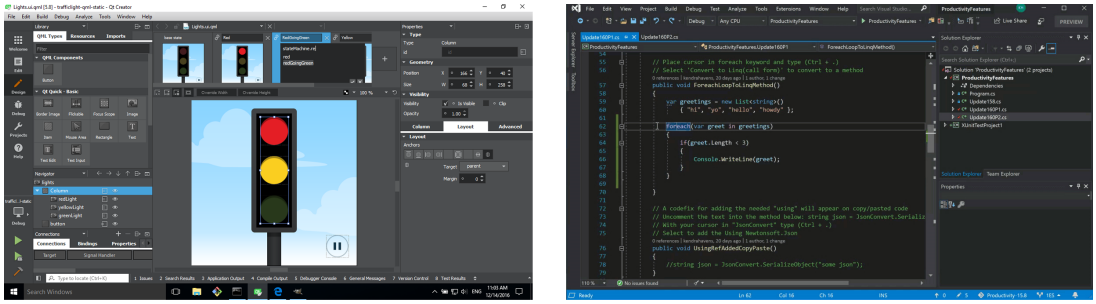
Niektoré kroky je môžné rozdeliť ešte na niekoľko ďalších krokov, pokiaľ sú veľmi zložité alebo rozsiahlé. Programátori alebo softvéroví inžinieri sa snažia o logické pochopenie a jednoduché (pokiaľ sa dá, tak aj elegantné) spracovanie všetkých problémov.

1.5 Pomôcky pre vývoj softvéru

Nástroje alebo pomôcky pre vývoj softvéru sú počítačové programy, ktoré využívame pri tvorbe softvérov pre tvorbu, debuggovanie (ladenie), údržbu a pod. Používame ich aj v rôznych štádiách softvérového vývoja, aby sme zväčšili efektívnosť, presnosť, osobný komfort, kvalitu a rýchlosť výsledného produktu.

1.5.1 IDE

Integrované vývojové prostredie alebo IDE (angl. Integrated Development Enviroment) je softvér poskytujúci rozsiahle integrované vybavenie pre programátorov. Obsahuje editor zdrojového kódu, kompilátor a debugger. IDE sa odlišne špecializujú pre konkrétny programovací jazyk, aby sa čo najviac zhodovali s formou a vzorom jazyka. Medzi známe IDE patria napríklad *Microsoft Visual Studio, Eclipse, CLion, PyCharm ...* .



Obrázok 2: Ukážka IDE, Anton Kica

1.5.2 Pamäťové debuggery

Pamäťové debuggery sú špeciálne debuggery, ktoré vyhľadávajú problémy únikov počítačovej pamäte, preplnenie vyrovnávaciej pamäte alebo fragmentácie pamäte. Pomáhajú predchádzať zvláštnemu a nevyspytateľnému správaniu softvéru počas jeho používania. Medzi známe pamäťové debuggery patria napríklad *Valgrind* alebo *WinDbg*.

1.5.3 Profiler

Profiler je program pre dynamickú analýzu programu, ktorý meria napríklad pamäťovú alebo časovú komplexnosť programu, využívanie konkrétnych inštrukcií alebo frekvenciu a trvanie funkcií. Informácie poskytnuté týmto programom sa ďalej využívajú pre optimalizáciu programu. Medzi známe profilery patria napríklad *gprof, Valgrind*.

1.5.4 Kontrola verzií

Program pre kontrolu verzií prehľadne spravuje zmeny v dokumentácii, počítačových softvéroch, internetových stránkach a pod. Tieto zmeny sú zväčša charakteristicky označované. Veľmi často po nich nasleduje číslo verzie napr. *ver. 1* a v ďalšiej verzii *ver. 2*. Tiež sú často asociované s časom vydania a osobou (autorom) danej verzie. Medzi najznámejší kontrolór verzií patrí *Git* so službou *GitHub*.

1.6 Počítačová grafika

Počítačová grafika je obor počítačovej informatiky, ktorý sa zaoberá metódami manipulácie a tvorby digitálnych obrazov. Využíva sa v oblastiach ako napríklad modelovanie, renderovanie, animácia, grafické uživateľské rozhranie (GUI), vizualizácia, spracovanie obrazu, 3D skenovanie, virtuálna realita a pod.

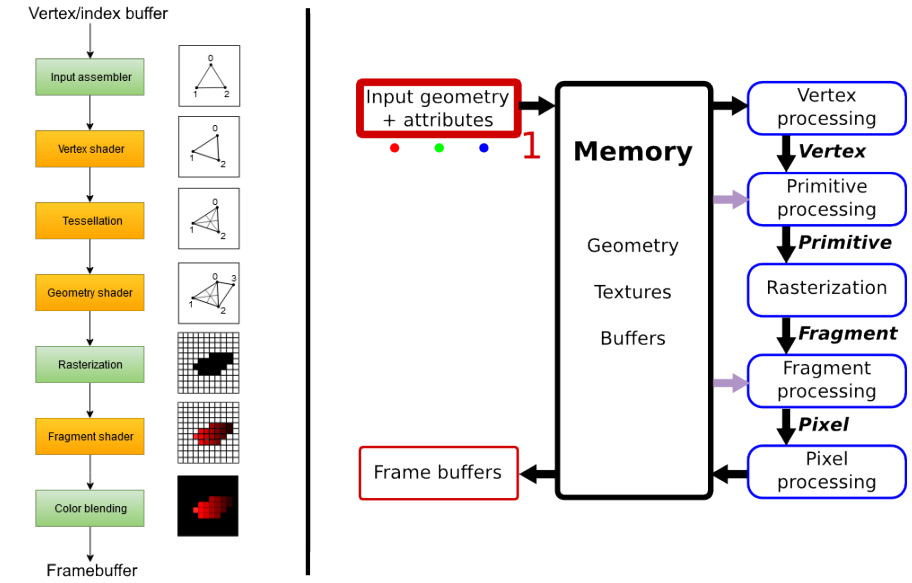
Grafika sa vyskytuje v mnohých priemysloch a to napr. v hernom priemysle, kreslených a animovaných filmoch, vizuálnych efektoch, medicíne, simuláciach a v mnohých ďalších priemyslových oblastiach.

1.6.1 Grafické API

Pri programovaní grafiky sa programátor často stretáva s grafickými API (Application Programing Interface), ktoré sú kolekciou funkcií vykonávajúcich sadu súvisiacich operácií. Medzi známe grafické API patria *OpenGL, Direct3D, Vulkan, OGRE*.

1.6.2 Grafický vykresľovací reťazec

Grafický vykresľovací reťazec alebo angl. *Graphics Pipeline* je špeciálny hardvérový/softérový podsystém, ktorý efektívne kreslí trojrozmerné primitívy (čiara, mnohouholník) do perspektívy. Základnou operáciou tohto vykresľovacieho reťazca je zmapovať 3D vrcholy do 2D priestoru na obrazovke. Tento proces je zložený z niekoľkých krokov, ktoré možno vidieť na obrázku:

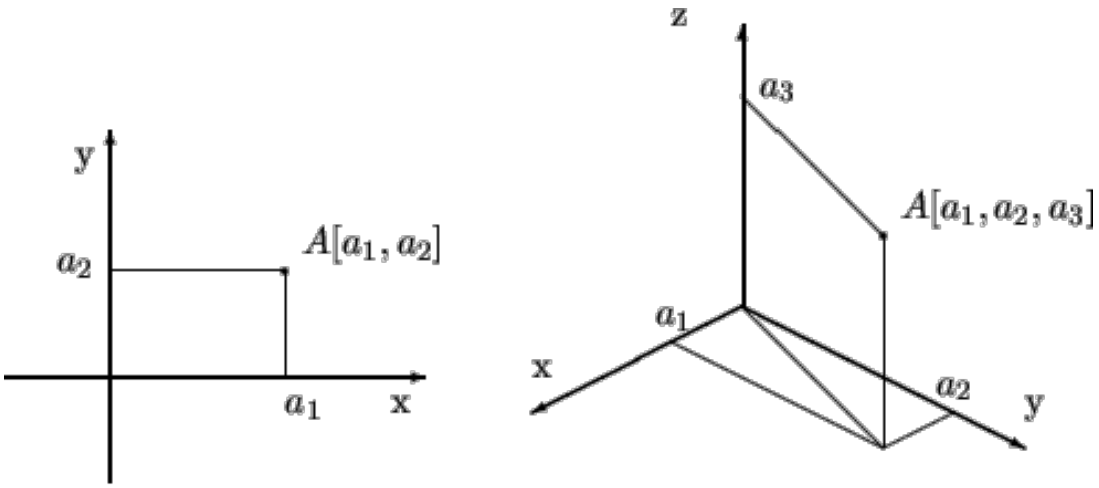


(https://vulkan-tutorial.com) (http://romain.vergne.free.fr)

Obrázok 3: Zjednodušený vykresľovací reťazec API Vulkan (vľavo) a OpenGL(vpravo)

1.6.3 Vektory

"Vektor je geometrický objekt, ktorý je určený dĺžkou, smerom a orientáciou. Môžeme si ho predstaviť ako orientovanú úsečku, t.j. úsečku, na ktorej je vyznačený začiatočný a koncový bod. Pritom nesmieme zabudnúť, že dve rôzne orientované úsečky, ktoré majú zhodnú dĺžku (t.j. veľkosť), smer aj orientáciu, predstavujú ten istý vektor, ide o dve rôzne umiestnenia toho istého vektora." [4]



(https://www.math.sk/skripta/node22.html)

Obrázok 4: Znázornenie vektorov v rovne a priestore

Vektory sú veľmi využívané vo všetkých oblastiach, kde je trojrozmerný priestor. Veľmi často sa využívajú ich vlastnosti pri rôznych vyjadreniach a výpočtoch.

1.6.4 Matice

"V matematike je matica (v mn. čísle matice) obdĺžnikový agregát čísiel, symbolov alebo vyjadrení zapísaných v riadkoch a stĺpcoch." [2] (preložené z angličtiny)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

(https://en.wikipedia.org/wiki/Matrix_(mathematics))

Obrázok 5: m×n Matica

Matice sa často využívajú na premenu geometrických dát do rozličných súradnicových systémov. Toto sa využíva napríklad pri kreslení pixelov na obrazovku. Obrazovka monitoru je iba dvojrozmerná a grafické API kreslia iba koordináty, ktoré sú normalizované (ich súradnice sú napr. v intervale <-1, 1>), ak nie sú normalizované, nenakreslia sa na obrazovku. Preto sa objekty z trojrozmerného priestoru vhodnou transformáciou matíc premietajú na dvojrozmernú plochu so svojou hĺbkou.

2. Ciele práce

Naším cieľom je vytvoriť funkčný softvér bez použitia cudzieho enginu a to takmer od základov. Chceme vytvoriť produkt, pomocou ktorého by sme dokázali simulovať cestnú dopravu. Náš produkt by mal ponúkať prostredie pre užívateľa, v ktorom môže vytvárať rôzne modelové situácie. Preto pre dosiahnutie tohto cieľa musíme:

- zvoliť vhodný hardvér a softvér, s ktorými budeme tento program vytvárať,
- navrhnúť a naprogramovať celú funkčnosť nášho softvéru,
- otestovať funkčnosť nášho produktu.

Po splnení týchto cieľov by sme mali byť schopný tvoriť rôzne cestné úseky, ktoré sú podobné reálnym cestným úsekom. Následne by sme na týchto vytvorených cestných úsekoch mohli vyhľadávať rôzne problematické časti a experimentovať s nimi pri nárhovaní vhodného riešenia.

3. Materiál a metodika

Pred samotnou prácou bolo pre nás nevyhnutné si premyslieť, zvoliť a po prípade vyhľadať a získať všetky prostiedky a technológie potrebné pri samotnej tvorbe softvéru.

3.1 Hardvér

Simulácie často obsahujú väčšie množstvo objektov, ktoré sa navzájom veľmi často ovplyvňujú, čo znamená, že je potrebné mať dostatok pamäte a aj pomerne výkonné komponenty počítača.

Predpokladali sme, že budeme potrebovať modernejší procesor s frekvenciou jadier 3.0 GHz a aspoň 4 jadrami, kedže procesor by mal vykonávať niektoré inštrukcie asynchrónne.

Grafika nášho programu nemala byť veľmi náročná. Preto sme predpokladali, že bude postačujúca grafická karta s frekvenciou jadier 1000 MHz a virtuálnou pamäťou 2GB.

RAM s frekvenciou 1600 MHz a kapacitou 8 GB bola tiež dostačujúca.

3.2 Softvér

Náš softvér by sa mal dať spustiť na veľkom množstve počítačov súčasnej doby. Veľmi rozšírený operačný systém je *Windows 10*, v ktorom sme aj vyvíjali náš softvér, tiež ho je možné aj na tomto operačnom systéme spustiť.

IDE, v ktorom sme písali, spracovávali a udržiavali kód, bol *Microsoft Visual Studio 2019*. *MSVC 2019* obsahuje veľké množstvo nástrojov na efektívnu činnosť, prehľadné a ľahké písanie kódu, jeho debuggovanie a kompilovanie.

Modelovací softvér *Blender* nám poslúžil na tvorbu modelov a ich textúr, ktoré sme použili v samotnej simulácii.

3.3 Programovací jazyk

Pri tvorbe programu sme použili programovací jazyk *C++*. Tento jazyk je veľmi rýchly, prehľadný, pomerne jednoduchý a poskytuje niekoľko programovacích paradigiem, ktoré boli pri vývoji veľmi vítané.

3.4 Pomocné knižníce a API

Pre šetrenie času a jednotnosť programovania sme použili niekoľko knižníc a API.

API

- *Vulkan* nám poskytol prístup k GPU a teda k veľkému množstvu funkcií, ktoré sme použili na kreslenie na obrazovku. Toto API je rýchle, intuitívne sa v ňom programuje, ale potrebuje na svoju činnosť dostať veľmi presné informácie.

Knižnice

- *GLFW* ponúka jednoduché vytvorenie okna, registrovanie eventov (udalostí) a vstupu užívateľa.
- *GLM* obsahuje užitočné matematické koncepty. Veľmi často sme využívali vektory a operácie s nimi a aj matice potrebné pre kreslenie grafiky.
- *ASSIMP* dokáže otvárať rôzne súbory modelov, vytvorené v Blenderi.
- *STB* dokáže otvárarať rôzne súbory textúr.
- *ImGUI* pomáha vytvoriť jednoduché a prehľadné GUI (Graphical User Interface).

Všetky zvolené prostriedky, sme využili pri navrhovaním a programovaním jednotlivých častí programu. Opísali sme postup vymýšlania a tvorby každej časti a všetko vytvorené sme nakoniec otestovali.

4. Výsledky práce

4.1 Tvorba softvéru

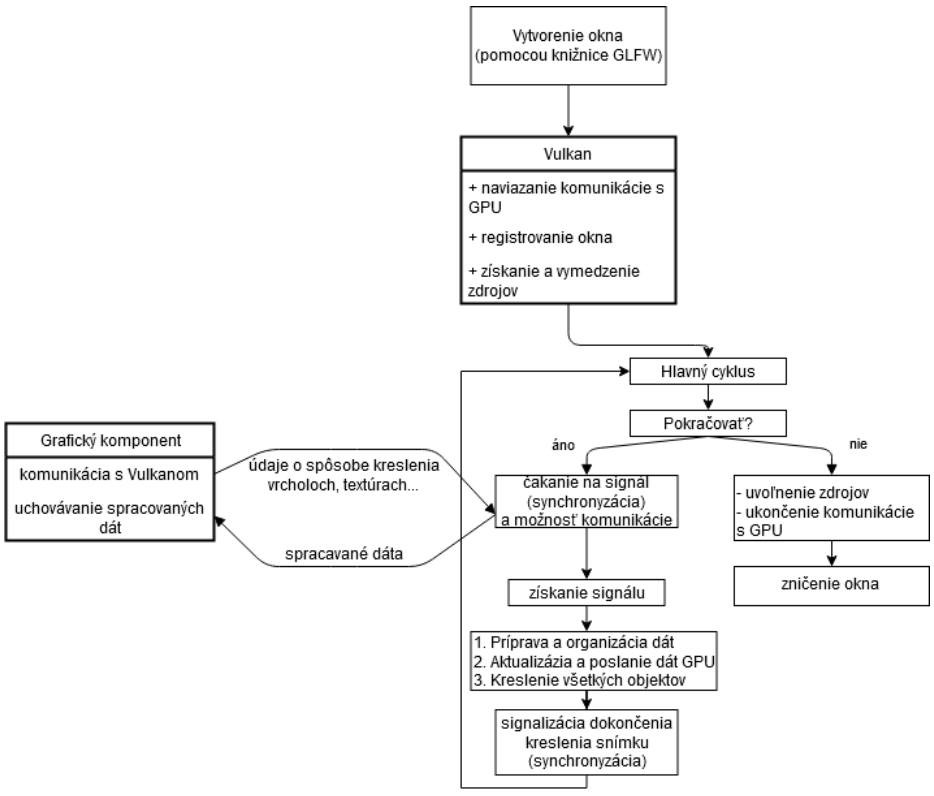
4.1.1 Grafika

Program, ktorý sme vytvorili, sme museli vytvárať takmer od základov. Toto pre nás znamenalo aj to, že sme potrebovali vymyslieť spôsob, ktorým by sme nakreslili rôzne objekty na obrazovku a to čo najvýhodnejším spôsobom.

Grafické API Vulkan poskytuje svojou veľkou abstrakciou veľa možných spôsobov, čo nám umožnilo veľkú flexibilitu pri navrhovaní, vývoji a použití nášho primitívneho grafického enginu.

Najprv sme vytvorili samotné okno, do ktorého sme kreslili grafiku, aby sme videli produkty našej simulácie. Potom sme nastavili, vytvorili a pripravili základnú funkčnú časť pre kreslenie pomocou Vulkanu. Ďalej sme vytvorili jednoduché rozhranie Vulkanu, prostredníctvom ktorého oznámime nášmu enginu, čo a ako chceme nakresliť cez rozhranie grafického komponentu.

Každý jeden cyklus grafického enginu používa doposiaľ poskytnuté dáta na nakreslenie nového snímku, ktorý sa zobrazí na obrazovku. Keď si bude užívateľ priať program ukončiť, engine ukončí kreslenie a uvoľní operačnému systému všetky použité prostriedky. Spôsob, akým tento engine funguje, vyzerá približne takto:

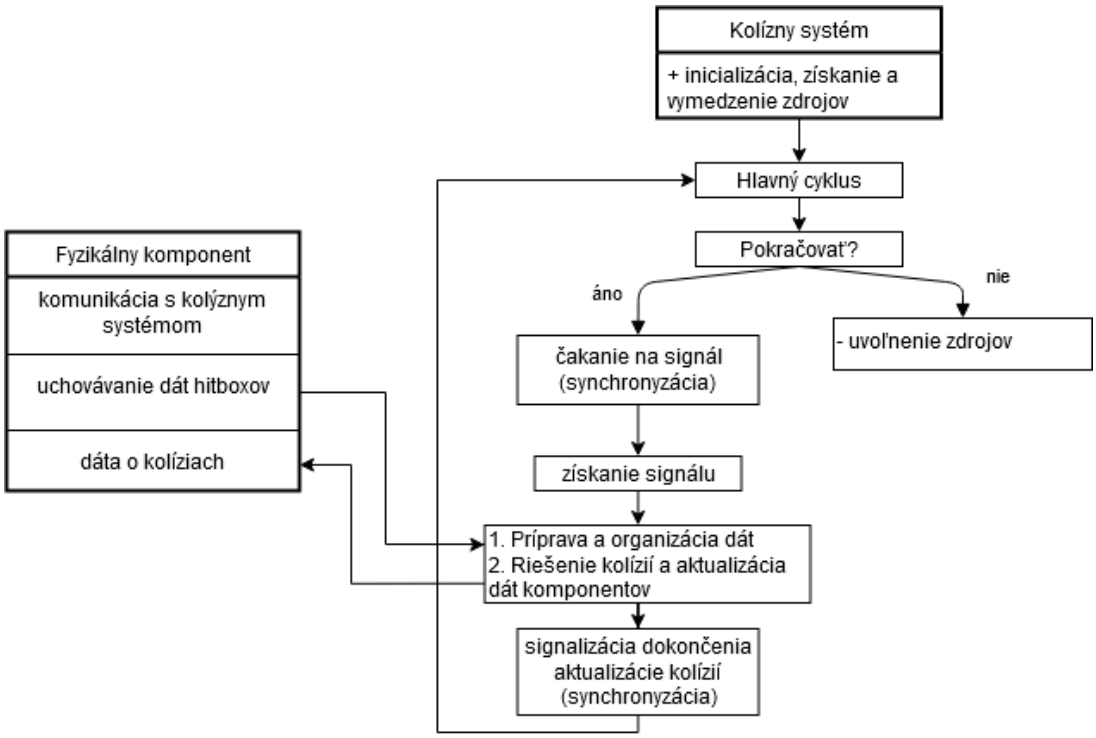


Obrázok 6: Diagram fungovania grafického enginu, Anton Kica

4.1.2 Hitboxy - kolízie

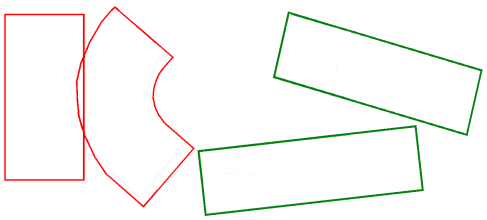
Vedeli sme, že cesty a najmä autá budú musieť mať medzi sebou istú formu interakcie - objekty musia vedieť svoju vzájomnú polohu, aby mohli adekvátne reagovať. Vzhľadom k tomu, že takýchto objektov mohol byť veľký počet, rozhodli sme sa vytvoriť jednoduchý systém na detekciu ich vzájomných polôh. Z tohto dôvodu sme zostrojili pseudokolízny systém s funkčnou jednotkou *hitbox*.

Spôsob, akým tento systém funguje, sme navrhli približne podobne, náš grafický engine, len s jedným väčším rozdielom. Grafický engine mal za úlohu kresliť grafiku, zato kolízny systém mal za úlohu riešiť kolízie.



Obrázok 7: Diagram fungovania kolízneho enginu, Anton Kica

Tento systém nám rýchlo a efektívne pomohol vyriešit nasledovné kolízie:



Obrázok 8: Znázornenie kolízie objektov, Anton Kica

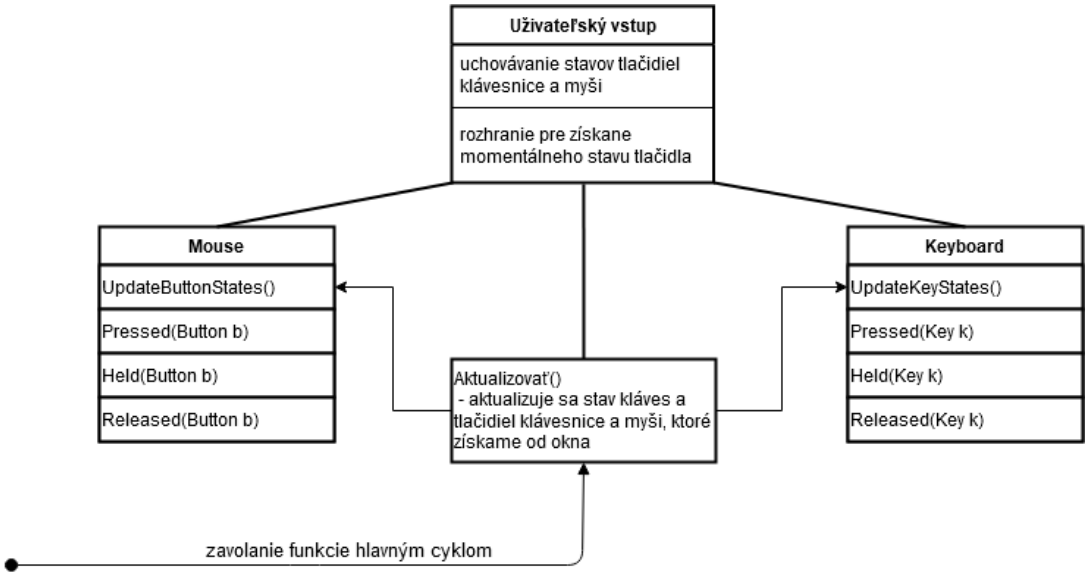
Červená farba znázorňuje objekty, ktorých hitboxy sú v kolízii, zelená farba znázorňuje hitboxy objektov, ktoré nie sú v kolízii.

4.1.3 Užívateľský vstup

Poslednú vec, ktorú sme museli vytvoriť pred samotnou tvorbou objektov simulácie, bol vstup užívateľa. Tvorba systému spracovania vstupu bola pomerne jednoduchá, pretože nám s tým nesmierne pomohla knižnica GLFW.

Okno, ktoré sme už vytvorili, tiež zaznamenáva rôzne vstupy klávesnice a myši ako napríklad stisnutie klávesy *'D'* alebo ľavého tlačidla myši. My sme museli iba túto informáciu vhodne spracovať a uchovať.

Počas každého jedného snímku sa zaznamenáva a posudzuje status každej klávesy a tlačidla myši. Stav ľubovoľného tlačidla je možné získať kedykoľvek je to potrebné. Systém spracovania vstupu vyzerá nasledovne:



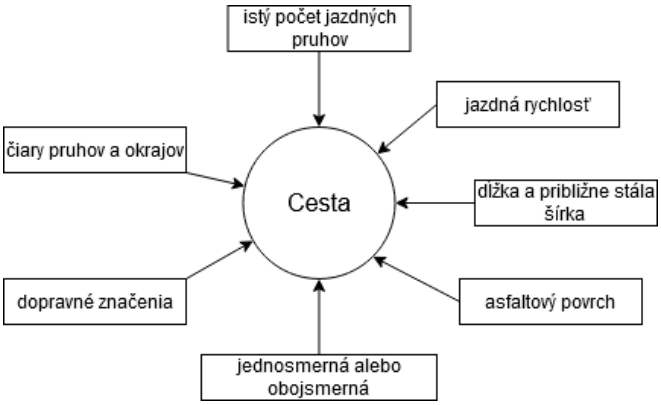
Obrázok 9: Znázornenie systému vstupu, Anton Kica

4.1.4 Objekty simulácie

Po tom, ako sme si všetky potrebné veci pripravili, sme začali tvorbou objektov, z ktorých bude pozostávať náš simulačný program.

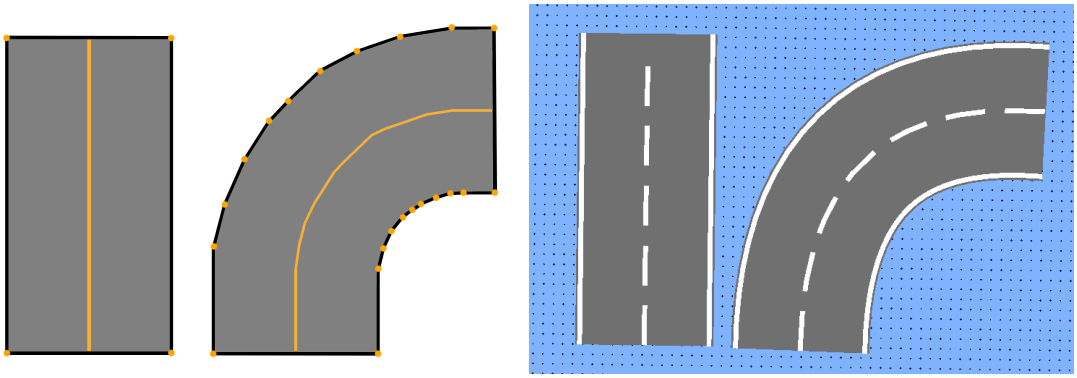
4.1.4.1 Cesta

Jedným zo základných častí simulácie sú cesty, pretože sa na nich budú pohybovať všetky autá. Cestu sme si charakterizovali približne takto:



Obrázok 10: Charakter cesty, Anton Kica

Tieto vlastnosti sme využili pri programovaní cesty. Najprv sme vytvorili objekt reprezentujúci cestu z predošlých definovaných vlastností, zkonštruovali sme cestu:



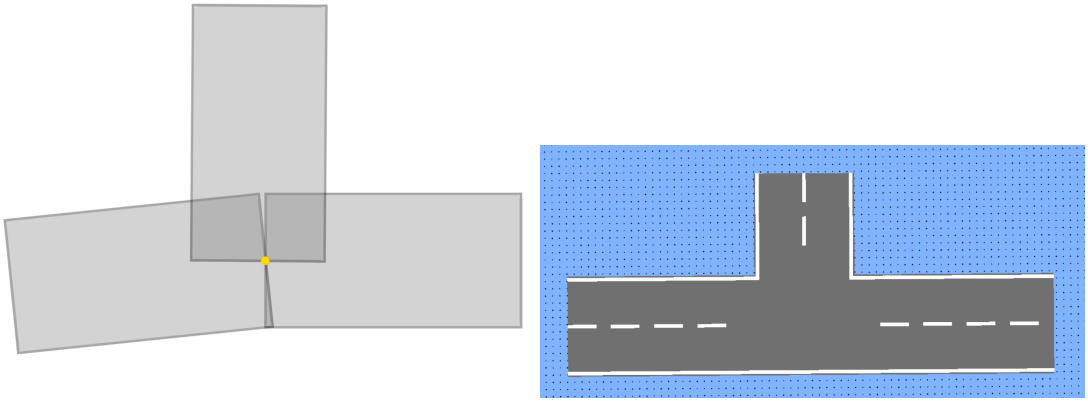
Obrázok 11: Návrh kostry, tvaru a realizácia cesty, Anton Kica

Navrhnutá cesta je osovo súmerná. Žlté body na ľavom a pravom okraji útvaru sú párové hraničné body, ktoré sú osovo súmerné podľa stredovej osi. Cesty sa dajú spájať na koncoch, čím sa môžu tvoriť cesty rôznych tvarov.

Keď už sme mali spravené tvary ciest, vytvorili sme na nich pruhy, po ktorých jazdia autá.

4.1.4.2 Križovatka

Ďalšou zo základných častí našej simulácie boli križovatky. Križovatky sme sa rozhodli zostrojiť ako aspoň tri do seba idúce cesty s jedným spoločným bodom. Naša predstava vyzerala približne takto:

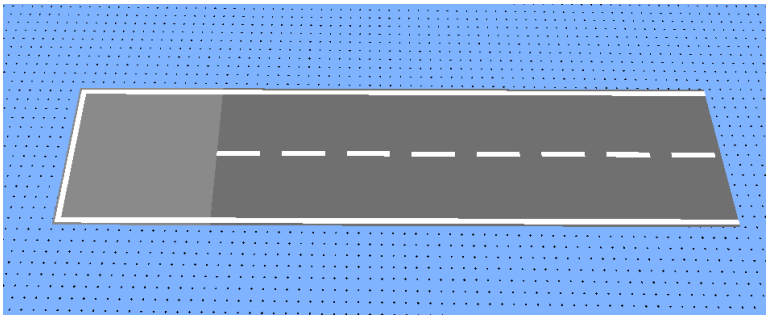


Obrázok 12: Návrh kostry, tvaru a realizácia križovatky, Anton Kica

Križovatky sú samostatné objekty, na ktoré sa pripájajú cesty - tie sú tiež samostatnými objektami. Ďalej sme ešte na križovatkách vytvorili pruhy, po ktorých jazdia autá.

4.1.4.3 Vstupy a výstupy áut

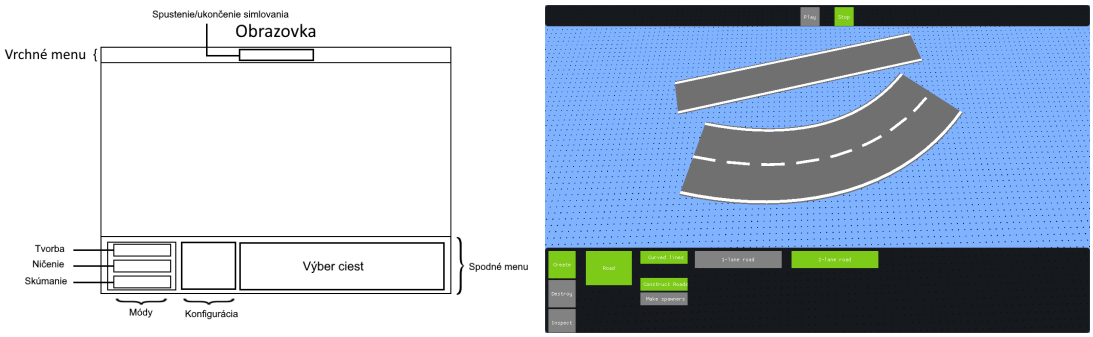
Autá sa museli na a aj z vytvorených ciest nejakým spôsobom dostať, preto sme zvolili metódu vstupov a výstupov áut. Tieto vstupy a výstupy sa pripájajú na voľné konce ciest. Vstupy majú za úlohu vytvárať autá a posielať ich k výstupom, kde jednoducho zmiznú, keďže splnili svoj zámer. Jeden objekt môže byť súčasne vstupom ale aj výstupom pre autá.



Obrázok 13: Vstupný a výstupný bod pre autá, Anton Kica

4.1.4.4 Užívateľské rozhranie

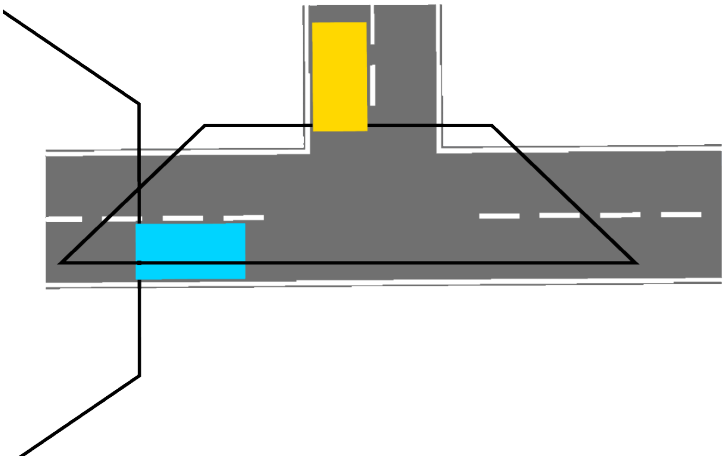
Už sme vytvorili *cesty*, *križovatky*, *vstupy* a *výstupy* a keďže sme sa snažili o prehľadnosť, tak sme vytvorili jednoduchú ponuku vytvárania týchto objektov pre užívateľa. Užívateľské rozhranie, ktoré sme vytvorili pomocou knižnice *ImGui*, sme navrhli takto:



Obrázok 14: Návrh a realizácia uživateľského rozhrania, Anton Kica

4.1.4.5 Autá

Tvorba áut nebola ničím výnimočná. Cieľom áut je sa dostať z jedného bodu do druhého bodu, pričom sa snažia dodržiavať isté pravidlá a nesnažia sa poškodiť resp. naraziť do druhých áut. Autá sme vytvorili približne rovnako, ako by sa správal a fungoval reálny človek za volantom, to znamená dávať si pozor pred seba a vedľa seba, zabáčať a dostať sa do svojho cieľa:



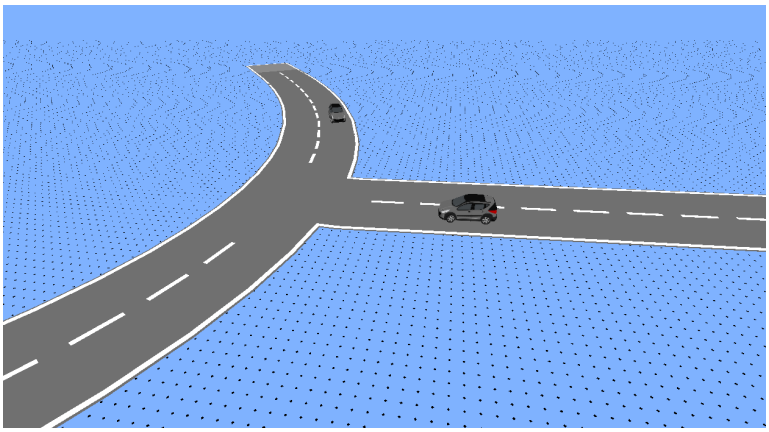
Obrázok 15: Jednoduché znázornenie áut, Anton Kica

Autá vidia okolo seba a vhodne reagujú na okolité podnety.

4.2 Finalizácia a výsledok

Všetky vytvorené časti sme zlúčili. Vytorili sme hlavný cyklus, ktorý všetko synchronizoval. Grafiku a kolízie sme naprogramovali na používanie na osobitných vláknach procesora. Vstup, aktualizovanie a reakcie objektov simulácie sme sa rozhodli spracovať v hlavnom cykle.

Po tomto všetkých krokoch potrebných pre vytvorenie programu sme si vytvorili skúšobnú trasu. Videli sme, že cesty, križovatky, vstupy, výstupy a autá fungujú.



Obrázok 16: Cesty, križovatky, vstupy, výstupy a autá, Anton Kica

Následne sme mohli vytvárať rôzne cestné úseky, ktoré sme mohli pozorovať a ľubovoľne upravovať. Vytvorili sme funkčný program na simulovanie dopravy.

5. Diskusia

Podarilo sa nám vytvoriť program na simulovanie cestnej dopravy od základov. Podarilo sa nám vytvoriť základné systémy - grafika, kolízie a užívateľský vstup, na ktorých sme neskôr mohli a aj začali vytvárať simulačné objekty - cesty, križovatky, vstupy, výstupy a autá. Zostavením týchto častí sme boli schopní spustiť program.

Po vytvorení ľubovoľného cestného úseku sme pozorovali priebeh cestnej dopravy. Pozorovali sme približne rovnaké správanie áut, s ktorým sa bežne stretávame v živote, Týmto môžme usúdiť, že program by sa dal využiť na simulovanie cestnej dopravy z reálneho života a tiež jeho pozorovanie alebo prípadne upravenie, ak by bolo potrebné riešiť konkrétny problém.

Musímé dodať, že sme boli prekvapení jeho málou zložitosťou, ale na druhú stranu jednoduchosť je niekedý vítaná. Tiež sme si pripravili veľmi dobré zázemie pre budúci vývoj nášho programu. Rozhodne by sme cheli pridať väčšiu flexibilitu ciest, vylepšiť tvorbu a prácu s cestami, interakciu a aj pridať prvky mesta. Je veľký počet možností a vylepšení, ktoré by sme chceli v budúcnosti zakomponovať do nášho programu., zatiaľ sme však s funkčnosťou nášho produktu spokojní.

6. Záver

Dokázali sme vytvoriť program, pomocou ktorého sme boli schopný do istej miery simulovať cestnú dopravu. Jednotlivé komponenty - jednoduchý grafický engine, kolízny systém, užívateľský vstup, cesty, križovatky, vstupné a výstupné miesta pre autá, uživateľské rozhranie a autá. Program zatiaľ obsahuje niekoľko prvkov, ktoré sa dajú využiť pre simuláciu dopravy.

Splnili sme všetky nami stanovenie cieľe a to aj primárny - vyvtoriť funkčný program, ktorý sa dá využiť na simulovanie rôznych dopravných simulácií.

7. Zhrnutie

Naším prvotným cieľom bolo vytvoriť simulačný program na simulovanie cestnej dopravy. Tvorba simulácie pozostávala z niekoľkých častí.

Najprv sme si museli zvoliť hardvér a k nemu všetok potrebný softvér, ktorý by sme neskôr využívali. Potom sme si zvolili programovací *C++* jazyk ako nástroj, s ktorým sme vytvárali náš program.

Po prípravnej časti sme začali uvažovať, navrhovať a programovať všetky časti a objekty potrebné pre chod nášho softvéru. Naprogramované časti sme nakoniec dali dokopy. Všetky časti spolu fungovali a vznikla prvotná verzia nášho programu. Program sme potom otestovali s vcelku pozitívnymi výsledkami, všetko fungovalo.

8. Prehľad použitej literatúry

[1] Algoritmus. URL https://sk.wikipedia.org/wiki/Algoritmus.

[2] Matice. URL https://en.wikipedia.org/wiki/Matrix_(mathematics).

[3] Stackoverflow. URL https://stackoverflow.com.

[4] Vektory. URL https://www.math.sk.

[5] Steve Marschner, Peter Shirley a kolektív. *Fundamentals of Computer Graphics*. A K Peters/CRC Press, 2015. ISBN 9781482229417.