ŠIAULIŲ UNIVERSITETAS
TECHNOLOGIJOS, FIZINIŲ IR BIOMEDICINOS MOKSLŲ FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Andrius Galmimas
Informatikos specialybės II kurso nuolatinių studijų studentas

Virtualios Saulės sistemos kūrimas ir analizė

Development and analysis of the Virtual Solar System

MAGISTRO DARBAS

Darbo vadovas:
Doc. K. Žilinskas
Recenzentas:
Doc. G. Felinskas

Šiauliai
2016 m.

„Tvirtinu jog darbe pateikta medžiaga nėra plagijuota ir paruošta naudojant literatūros sąraše pateiktus informacinius šaltinius bei savo tyrimų duomenis“

Darbo Autorius _____

(vardas, pavardė, parašas)

Turinys

Darbo tikslas ir uždaviniai

Darbo tikslas – Sukurti virtualų Saulės sistemos modelį ir atlikti jo analizę.

Uždaviniai

- Sudaryti Saulės Sistemos matematinį modelį, kuris galėtų būti realizuojamas bet kuriam grafiniam varikliui;
- Pritaikyti matematinį modelį pasirinktam grafiniam varikliui;
- Sukurtoje programoje vartotojas galėtų:
    - Gauti informaciją apie Saulės Sistemą;
    - Stebėti kaip Saulės Sistemos kūnus veikia įvestas papildomas kūnas su pasirenkamais parametrais;
    - Kurti savo virtualią Žvaigždės Sistemą.
- Realizuoti matematinį modelį keliems grafiniams varikliams;

Darbo Vadovo _____

(vardas, pavardė, parašas)

Įvadas

Pirmosios astronomijos žinios pradėtos kaupti nuo seniausių laikų, pirkliai ir jūrininkai orientuodamiesi žvaigždžių padėtimi nustatydavo savo būvimo vietą ir kelionės tikslą keliaudami karavanas per dykumas ar plaukdami laivais. Sekdami Mėnulio fazes, žemdirbiai, medžiotojai, žvejai išmoko matuoti laiką pagal žvaigždžių išsidėstymą ir taip nustatydavo sėjos pradžią, nuspėdavo potvynius arba derliaus nuėmimo metą. Pagal žvaigždžių išsidėstymą danguje jie suskirstė jas į žvaigždynus, bandė suprasti ir paaiškinti dangaus reiškinius.

Senovės egiptiečiai, babiloniečiai, finikiečiai, kinai ir majai (3000 m. pr. m. e.) ir kitos senovės civilizacijos kaupė ir sistematizavo astronomijos žinias. Apie 4000 m. pr. m. e. egiptiečiai sudarė saulės kalendorių (lot. *Calendae, Kalendae*) ir gana tiksliai nustatė metų trukmę. Babilonijoje 721 m. pr. m. e. aprašytas mėnulio užtemimas ir iš ten iki mūsų laikų išlikęs paros dalijimas į 24 valandas.

1609 m. išradus teleskopą, astronomijos raida paspartėjo. Galilėjo Galilėjus savo darbo teleskopu stebėjo Saulę, Mėnulį, Venerą, Jupiterį ir kitus dangaus kūnus. 1687 m. I. Niutonas suformulavo visuotinės traukos dėsnį, M. Lomonosovas aptiko, kad Venera turi atmosferą, 1781 m. V. Heršelis atrado Urano planetą.

Tobulėjant technologijoms ir atsiradus kompiuteriams, atsirado galimybė Saulės sistemą ir visą visatą simuliuoti virtualioje aplinkoje. naudojant žinomus dangaus kūnų judėjimo ypatybes. Tam labai pasitarnauja grafiniai varikliai.

Grafinis variklis – programinės įrangos sistema, kuri leidžia kurti 2D ir 3D grafika. Jis plačiai naudojamas kuriant kompiuterinius žaidimus.

Nors yra sukurta nemažai virtualių Saulės Sistemos modelių simuliacijų, daugelyje jų yra naudojamos apskaičiuotos nekeičiamos orbitos.

Šiame darbe pavaizduota kaip galima sudaryti Saulės ar bet kokia kitą žvaigždės sistemą naudojant Niutono visuotinės traukos dėsnį, mechanikos žinias ir skaitinius metodus. Modelis bus pritaikytas keliems grafiniams varikliams.

1. Teorinė dalis

1.1 Masės centras

Masės centras yra sutelktų jėgų centrinis taškas, per kurį eina kūno dalių sunkio jėgų atstojamoji. Masės centras yra naudingas atskaitos taškas mechanikos skaičiavimuose, kurie apima mases, pasiskirsčiusias erdvėje. Sąvoką svorio centras (masės centras) pirmasis pavartojo Archimedas, ji teigė, jog tai taškas, už kurio pakabintas kūnas yra pusiausviras.

1.1.1 Dalelių sistemos apibrėžimas

Kai sistema yra sudaryta iš dalelių r_i ($i = 1, \ldots, n$), kiekviena turi masę m_i, kuri yra erdvėje su koordinatėmis $\mathbf{r}_i$, ($i = 1, \ldots, n$), centro koordinatės **R** tenkina sąlygą

$$\sum_{i=1}^{n} m_i(\mathbf{r}_i - \mathbf{R}) = 0.$$

Išsprendus lygtį **R** atžvilgiu gauname formulę

$$\mathbf{R} = \frac{1}{M}\sum_{i=1}^{n} m_i\mathbf{r}_i,$$

kur M yra visų dalelių masių suma. [1]

1.2 Judėjimo reliatyvumas

Jei skirtingų atskaitos sistemų atžvilgiu kūnų padėtis ir judėjimas yra skirtingi, sakoma, kad padėtis ir judėjimas yra reliatyvūs.

Klasikinėje mechanikoje galioja mechaninis reliatyvumo principas: mechanikos dėsniai vienodi visose inercinėse atskaitos sistemose. Vadinasi, bet kokie eksperimentai, atlikti uždaroje sistemoje, neleidžia nustatyti, ar ši sistema kokio nors atskaitos taško atžvilgiu juda tiesiai ir tolygiai, ar yra rimties būvio.

Specialiąją reliatyvumo teoriją sukūrė Albertas Einšteinas. Jis parodė, kad koordinatės ir laikas įvairiose atskaitos sistemose yra susiję pagal Lorenco, o ne Galilėjaus transformaciją. Tačiau, esant mažiems greičiams, Lorenco transformacijų sistema tampa Galilėjaus transformacijų sistema.

Galilėjaus transformacija yra naudojama pereinant iš atskaitos sistemos $K\,(x, y, z, t)$ į kita – $K'\,(x', y', z', t')$. Atskaitos sistema K' juda pastoviu greičiu K atskaitos sistemos atžvilgiu. Galilėjaus transformacijos pagrįstos dviem aksiomomis:

- laiko tarpas tarp dviejų įvykių vienodas visose inercinėse atskaito sistemose;
- kūno matmenys nepriklauso nuo judėjimo greičio.

Turime dvi inercines atskaitos sistemas K ir K', ir pradiniu laiko momentu $(t = t' = 0)$ jų koordinačių pradžios sutampa. Tuomet Galilėjaus transformacija tokia:

$$x' = x - V_x t, \qquad y' = y - V_y t, \qquad z' = z - V_z t.$$

Laikas abiejose atskaitos sistemose eina vienodai $t = t'$ ir $\vec{r}' = \vec{r} - \vec{V}t$, čia x, y, z, x', y', z' – taško M koordinatės atskaitos sistemose K ir K' atitinkamai laiko momentu $t = t', \vec{r}, \vec{r}'$ to paties taško spinduliai vektoriai tose pačiose atskaitos sistemose. $V_x, V_y, V_z -$ atskaitos sistemos K' greičio V projekcijos į K ašis.

Dviejų taškų tarpusavio padėtis ir jų judėjimo greitis vienas kito atžvilgiu nepriklauso nuo atskaitos sistemos pasirinkimo, t.y. šie dydžiai Galilėjaus transformacijų atžvilgiu yra invariantiški. Jėga taip pat yra invariantiška Galilėjaus transformacijų atžvilgiu, t. y. $F = F'$. [1]

1.3 Visuotinės traukos dėsnis

Niutono visuotinės traukos dėsnis sako, kad bet kokie du Visatos kūnai veikia vienas kitą jėga, kurios modulis yra:

$$F = G\frac{m_1 m_2}{r^2};$$

Du kūnai vienas kitą tik traukia, todėl

$$F = G\frac{m_1 m_2 \vec{r}}{r^3};$$

Čia F – jėga, $m_1$ir m_2 – sąveikaujančių kūnų masės, r – atstumas tarp kūnų,

$G = 6,672 \cdot 10^{-11}\frac{Nm^2}{kg^2}$ – gravitacijos konstanta.

Dėsnis buvo atrastas 1687 m., Niutonas žinojo pagrindinį šio dėsnio trūkumą, nes pagal jo suformuluotą teoriją jėga veikia akimirksniu. Dabar žinoma, kad ribinis greitis yra šviesos greitis ir masyvūs kūnai iškreipia erdvėlaikį. Be to, Niutonas negalėjo paaiškinti, kodėl dėsnis yra būtent toks.

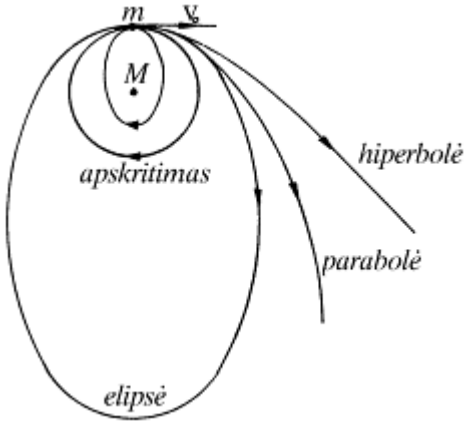
Gravitacija – sąveika tarp materialių kūnų, priklausanti nuo jų masės.

Gravitacinis centras – centras, kurios jėga traukia kokį nors kosminį kūną. Tas gravitacinis centras gali būti žvaigždė, galaktikos centras, planeta. Bet koks kosminis objektas yra veikiamas gravitacinio centro jėgos. [2]

1.4 Dviejų kūnų uždavinys. Apibendrinantieji Keplerio dėsniai

Niutonas, pasinaudojęs Keplerio nustatytais empiriniais planetų judėjimo dėsniais, išvedė visuotinės traukos dėsnį, o paskui išsprendė ir atvirkščią uždavinį: iš visuotinės traukos dėsnio išvedė visus tris Keplerio dėsnius, tiktai jau bendresniu pavidalu. Niutonas nagrinėjo dviejų kūnų (pavyzdžiui, Saulės ir planetos) judėjimą, veikiant tik jų tarpusavio traukos jėgai ir neįskaitė kitų kūnų (pavyzdžiui, kitų planetų) traukos įtakos. Kitaip tariant, Niutonas sprendė vadinamąjį **dviejų kūnų uždavinį**.

Tarkime, dviejų kūnų (tiksliau - materialiųjų taškų) masės yra M ir m, o atstumas tarp jų - r. Paprastai yra nagrinėjamas ne absoliutinis kūnų judėjimas, bet santykinis, t.y. vieno kūno judėjimas kito kūno atžvilgiu. Masės M kūnas (jį laikome centriniu) traukia masės m kūną jėga $-GMm / r^2$ ir suteikia jam pagreitį $-GM / r^2$. Savo ruožtu tokio pat dydžio, bet priešingos krypties jėga masės m kūnas traukia masės M kūną ir suteikia pastarajam pagreitį Gm / r^2. Tada masės m kūno pagreitis masės M kūno atžvilgiu yra $-GM / r^2 - Gm / r^2 = -G(M + m) / r^2$. Masės m kūno santykinio pagreičio projekcijos į stačiakampės koordinačių sistemos, kurios pradžia sutampa su masės M kūnu, ašis duoda tris antrosios eilės diferencialines lygtis, apibūdinančias santykinį judėjimą [2]:



$$\frac{d^2x}{dt^2} = -G(M+m)\frac{x}{r^3};$$
$$\frac{d^2y}{dt^2} = -G(M+m)\frac{y}{r^3};$$
$$\frac{d^2z}{dt^2} = -G(M+m)\frac{z}{r^3}.$$

1 pav. Dviejų kūnų uždavinys

Šio tipo lygčių sprendimas (integravimas) nagrinėjamas teorinės mechanikos kursuose. [2]

1.5 Trijų ir n (daugelio) kūnų uždavinys

Norint nustatyti, kaip juda trys kūnai, traukiantys vienas kitą jėga, atvirkščiai proporcinga atstumo tarp jų kvadratui, reikia spręsti uždavinį, vadinamą **trijų kūnų uždaviniu**. Trijų kūnų uždavinyje kūnų judėjimą išreiškia devynios ($3n$) antrosios eilės diferencialinės lygtys. Jas spręsti labai sunku ir bendru atveju iki šiol nepavyko išvesti formulių, kurios aiškiai aprašytų visų trijų kūnų judėjimo trajektorijos formą ir leistų numatyti jų padėtį erdvėje bet kuriuo laiko momentu. Tačiau yra atvejų, kai trijų kūnų uždavinys gali būti tiksliai išspręstas. Juos 1772 m. nurodė prancūzų matematikas ir mechanikas Žozefas Lagranžas (*J. L. Lagrange*). Šiais atvejais trečiasis kūnas turi būti viename iš penkių taškų, kurie vadinami **libracijos** (lot. libratio - svyravimas), arba **Lagranžo, taškais**. Trys Lagranžo taškai L_1, L_2 ir L_3 yra tiesėje, kuri eina per pirmuosius du kūnus (m_1 ir m_2); L_1 yra tarp tų kūnų, L_2 - į vieną pusę ir L_3 - į kitą pusę nuo jų. Konkrečios šių taškų padėtys priklauso nuo kūnų masių santykio. Taškai L_4 ir L_5 su kūnais m_1 ir m_2 sudaro lygiakraščius trikampius. Taigi *jeigu trečiasis kūnas yra kuriame nors Lagranžo taške, tai visi trys kūnai juda plokštumoje (kurioje jie yra) vienodos rūšies antrosios eilės kreivėmis, o santykiai tarp jų tarpusavio atstumų visą laiką lieka nepakitę*.

Astronomijoje ir kosmonautikoje didelę reikšmę turi supaprastintas trijų kūnų uždavinys, kai trečiojo kūno masė yra labai maža ir jis neturi įtakos kitų dviejų kūnų skriejimui aplink bendrą jų masių centrą.

Aišku, daugelio (n) kūnų uždavinys yra dar sudėtingesnis negu trijų kūnų uždavinys. Čia apytikslis vaizdas apie painų pasirinktojo kūno judėjimą gaunamas nuoseklaus artėjimo metodu, atskirai įvertinant kiekvieno kito kūno traukos jėgų poveikį jam. Tokiu pat būdu tikslinamos ir Saulės sistemos kūnų orbitos, nes Saulės sistema - tai taip pat daugelio kūnų sistema. [2]

Trijų kūnų uždavinys buvo nagrinėtas Z.E. Musielak‘o ir B. Quarles moksliniame straipsnyje „*The three-body problem*” [3]. Straipsnyje yra aprašoma trijų kūnų uždavinio problema, skirtingi analiziniai ir skaitinių metodų spendimo būdai. Nors straipsnyje yra aprašyta nemažai trijų kūnų uždavinio spendimo būdų, aš jais nesinaudoju, nes aprašyti metodų rezultatai yra apskaičiuotos ir nekeičiamos planetų skriejimo orbitos.

2 Projektinė dalis

2.1 Grafinių variklių pasirinkimo kriterijai

1. Pasirinktas grafinis variklis turi būti nemokamas ir prieinamas.
2. Turi pagalbines bibliotekas;
3. Kūrimo ir valdymo lengvumas.
4. Su grafiniu varikliu sukurta aplikacija turi veikti keliuose platformose, kaip Android, Mac ir pan.
5. Galimybė paleisti sukurtus projektus senesnėse OS.

2.2 Programinių įrankių ir priemonių analizė

2.2.1 HTML5

HTML5 yra HTML (*Hyper text Markup Language* „Hiperteksto žymėjimo kalba“) naujausia atmaina, kuri naudojama kurti ir pateikti žiniatinklio (WEB) puslapių turinį.

HTML5 buvo kuriamas išspęsti praeitų HTML versijų suderinamumo problemas. Vienas iš didžiausių skirtumų tarp HTML5 ir kitų jos kalbos versijų yra ta, kad senosios versijos reikalauja užpatentuotų papildymų ir API (Application Programming Interface „Aplikacijų programavimo sąsaja“), kad tinklaraščiai tinkamai butų atvaizduoti kaip dizaineris norėjo. HTML5 suteikia vieną bendrą sąsają kuri palengvina elementų įkėlimą.(Pavyzdžiui, nereikia įdiegti Flash papildinio į HTML5 nes elementas veiks pats savaime.).

Vienas iš HTML5 projekto tikslų yra palaikyti multimediją (tekstas, vaizdas, animacija, garsai, grafika bei kita kompiuterinė informacija) mobiliuosiuose įrenginiuose.

Daromai HTML5 Saulės sistemai reikės **Three.min.js** bibliotekos [4]. **Canvas** bibliotekoje yra apipavidalintas, todėl užtenka ją iškviesti panaudojus <head> arba <body> žymas, kurių vaizdavimo dydis yra redaguojamas CSS faile **index1.css**. Naudojama kalba objektų sudarymui yra Javascript.

Rekomenduojamos naršyklės:

- Internet Explorer 9 ir 10;
- Firefox 7 arba naujėsnės versijos;
- Chrome 14 arba naujėsnės versijos;
- Safari 5 arba naujėsnės versijos;

- Opera 11 arba naujėsnės versijos;
- Mobile Safari 3.2 arba naujėsnės versijos;
- Opera Mobile 5 arba naujėsnės versijos;
- Android 2.1 arba naujėsnės versijos;

Nors pats HTML5 turėtų veikti bet kuriame kompiuteryje yra netiesioginė rekomendacija jei naudojama Windows XP operacinė sistema, turi būti suinstaliuota SP2 (Service Pack 2) ir turėtų bent 256 RAM‘u atminties.

2.2.2 OpenGL

OpenGL (Open Graphics Library) yra standartinė kompiuterio aplikacinė programavimo sąsaja (API), kuri leidžia daugialypės terpės programoms (pvz., žaidimams, filmams ir grafikos dizaino programoms) veikti su aparatūra, kuri siunčia animaciją į kompiuterio ekraną. Prieš atsirandant *OpenGL*, bet kuri kompanija kurianti grafinės programas turėdavo perrašyti vaizdinę dalį kiekvienai operacinės sistemos platformai ir gerai žinoti grafinę aparatūrą. Su *OpenGL*, aplikacija gali atlikti tuos pačius veiksmus bet kurioje operacinėje sistemoje kuri naudoja *OpenGL* palaikantį vaizdo adapterį.

OpenGL apibūdina komandų seką arba iš karto veikiančias funkcijas ir operacijas. Kiekviena komanda nurodo piešimo veiksmą arba suteikia tam tikrus efektus, veiksmus. Sąrašas šių komandų gali buti sukurtos ir panaudotos veiksmų pasikartojimui. *OpenGL* yra nepriklausomas nuo lango charakteristikos nuo kiekvienos operacinės sistemos, bet suteikia specialius "glue" rutinas kiekvienai operacinei sistemai kuri leidžia OpenGL’ui veikti tos sistemos lango aplinkoje. OpenGL turi daug savyje įtaisytų galimybių kurios iškviečiamos per API, kaip paslėpto paviršiaus pašalinimas, tekstūros suliejimas, *OpenGL* operacijos, transformacijų modeliavimas ir rodymas, atmosferiniai efektai.

Buvęs kompiuterių techninės ir programinės įrangos gamintojas Silicon Graphics yra *OpenGL* pradininkas. [5]

OpenGL yra nemokamas ir prieinamas visiems kurti ir platinti juo sukurtas aplikacijas. Microsoft siūlo nemokamus bibliotekų parsisiuntimus Windows platformoms. Nors pats *OpenGL* nėra programavimo įrankių paketas, tokie paketai (*toolkit*) yra prieinami kaip Open Inventor, *GLUT, GLFW, OpenTK, OpenGL Utility*.

Didžiulė programavimo įrankių dalis palaiko *OpenGL* kaip Microsoft Visual Studio, CodeWarrior, Xcode, Intel c++ compiler.

OpenGL yra naudojamas visuose dabartinėse žaidimų kūrimo programose.

Projekte yra naudojama *Glut32* [6], *Tao.OpenGl* [7] ir *ShadowEngine*.

2.2.3 Unity

Unity yra multiplatforminis žaidimų kūrimo variklis sukurtas „*Unity* Technologies“ kompanijos ir yra naudojamas kurti video žaidimus PC, konsolėms, mobiliems įrenginiams ir tinklapiams. Pirmoji versija kuri buvo paskelbta tik „OS X“ operacinei sistemai 2005 metais, jis buvo vėliau išplėstas ir varikliu naudojasi daugiau nei penkiolika kito tipažo platformų.

Unity turi 2 leidimus:

1. Personal Edition, kuris yra nemokamas ir visiems prieinamas, bet metinė pajamų įplauka negali siekti 100 000 US, kitu atveju reikia pirkti profesionaliąja versiją arba bus taikomi autorinių teisių apribojimai.
2. Professional Edition, kurį galima naudoti tik mokant mėnesinį mokestį.

Daugiau informacijos yra pateikta *Priedas 2* ***Lentelė 5***.

Grafinio variklio redaktoriaus (editor) reikalavimai:

OS: Windows 7 SP1+, 8, 10; Mac OS X 10.8+.

Windows XP ir Vista yra nepalaikomi.

GPU: Vaizdo plokštė su DX9 (shader model 2.0) galimybėmis. Kūrėju teigimu viskas kas pagamintas nuo 2004 turėtu veikti. Kitkas priklauso nuo projekto sudėtingumo.

Papildomi platformų plėtros reikalavimai:

- iOS: Mac kompiuteris veikiantis OS X 10.9.4 versija ir Xcode 6.x.
- Android: Android SDK ir Java Development Kit (JDK).
- Windows 8.1 Store Apps / Windows Phone 8.1: 64 bit Windows 8.1 Pro ir Visual Studio 2013 Update 2+.
- WebGL: Mac OS X 10.8+ arba Windows 7 SP1+ (64-bit redaktoriuje tiktai).

Paprastai sukompiliuoti *Unity* žaidimai gali veikti bet kurioje sistemoje, tačiau veikimo kokybė priklauso nuo pačio projekto sudėtingumo.

Rekomenduojami reikalavimai:

- Darbalaukis:
    - OS: Windows XP SP2+, Mac OS X 10.8+, Ubuntu 12.04+, SteamOS+
    - Vaizdo plokštė: DX9 (shader model 2.0) galimybėmis. Kūrėju teigimu viskas kas pagamintas nuo 2004 turėtu veikti. Kitkas priklauso nuo projekto sudėtingumo.
    - CPU: SSE2 instruction set palaikymas.
    - Web Player: Reikalinga naršyklė kuri palaiko papildymus kaip IE, Safari ar Firefox.
- iOS: reikalauja iOS 6.0 operacinės sistemos arba vėlesnės.
- Android: OS 2.3.1 arba vėlenės; ARMv7 (Cortex) CPU su NEON palaikymu arba Atom CPU.
- OpenGL ES 2.0 arba vėlesnės versijos.
- WebGL: Bet kokios naujausios versijos naršyklės kaip Firefox, Chrome, Edge arba Safari.
- Windows Phone: 8.1 arba vėlesnės versijos.
- Windows Store Apps: 8.1 vėlesnės versijos.

2.2.4 Unreal Engine

Unreal Engine – žaidimo variklis, sukurtas įmonės „Epic Games“. Pirmąkart atvaizduotas 1998 m. pirmojo asmens šaudyklėje „Unreal“, nuo to laiko jis tapo pagrindu tokiems žaidimams kaip „Unreal Tournament“ serija, „Gears of War“, „BioShock“ ir t. t. Nors iš pradžių kurtas pirmojo asmens šaudyklėms, variklis buvo sėkmingai panaudotas ir daugelyje kitokio žanro kompiuterinių žaidimų, įskaitant stealth, MMORPG ir RPG.

Variklio kodas buvo parašytas C++ kalba, jis pasižymi dideliu portatyvumu, šiandieną juo naudojasi daugelis žaidimų kūrėjų.

UE4 yra skritas Microsoft DirectX 10–12 (Microsoft Windows, Xbox One, Windows RT), taip pat OpenGL (Mac OS X, Linux, PlayStation 4, iOS, Android), ir JavaScript/WebGL (interneto naršyklės su HTML5).

Viena iš UE4 pagrindinių savybių yra *Blueprint* vizuali programavimo sistema, kur vartotojas gali tiesiogiai stebėti sudėliotą programos veikimą, čia žaidimų programuotojai ir dizaineriai gauna reikiamus įrankius kurti ir modeliuoti žaidimo eigą be būtinybės rašyti programavimo kodą.

Kad veiktų visos grafinio žaidimo variklio redaktoriaus savybės, reikia naudojamame PC arba Mac įdiegta *Visual Studio 2015 C++*, ir *UE4* variklio versija turi būti **4.10** arba naujesnė.

Rekomenduojami minimalus reikalavimai grafinio variklio redaktoriui:

- PC arba Mac;
- Windows 7, 8, 10 64-bit arba Mac OS X 10.9.2 arba vėlesnės versijos;
- Quad-core Intel arba AMD procesorius, 2.5 GHz arba greitesnis;
- NVIDIA GeForce 470 GTX arba AMD Radeon 6870 HD vaizdo plokštė arba naujesnė;
- 8 GB RAM

Rekomenduojami reikalavimai grafinio variklio redaktoriui:

- Intel Core i7 4930K procesorius;
- Intel X79 motininė plokštė;
- 32GB RAM;
- 1TB SSD kietasis diskas;
- Nvidia GTX 770 vaizdo plokštė.

2.3 Pasirinktų grafinių variklių palyginimas

1 lentelė. Grafinių variklių lyginimas

Nr.	Paaiškinimas	HTML5 (Javaskript)	OpenGL (C#)	Unity (C#)	UE4 (C++), Blueprint
1.	Yra nemokamas	+	+	+/-	+
2.	Realaus laiko klaidų paieška	-	+	+	+
3.	Funkcijų ir operatorių pavyzdžiai	-	+	+	+
4.	Matematinis paketo iškvietimas	Math.PI	Math.PI	Mathf.PI	PI
5.	Standartiniai ciklai (for, while)	+	+	+	+
6.	Standartiniai sąlygos sakiniai (if, if... else)	+	+	+	+/-[1]
7.	3D Vektoriaus apipavidalinimas variklyje	THREE.Vector3()	Vector3()	Position2()[2]	FVector()
8.	Integruota pagalbos biblioteka	-	+/-[3]	+	+
9.	Kodo vykdymas darbiniame lauke	-	+	+	+
10.	Veikia daugelyje platformų	+	+	+	+
11.	Veikia senesniuose nei 2004 m. kompiuteriuose	+	+/-[4]	+/-[5]	-
12.	Atskirų, papildomų įrankių įdiegimas	-	+	+/-	+/-

1. Jei naudojama *Blueprint* sistema, standartinis sąlygos sakinys yra ***branch***.
2. Sudaroma vektorių apipavidalinanti struktūra.
3. Priklauso nuo naudojamo paketo.
4. Vaizdo plokštė turi palaikyti *OpenGL*.
5. Veikimas priklauso nuo programos sudėtingumo.

2.4 Pasirinktas pagrindinis grafinis variklis

Pagrindinės priežastys kodėl *Unity* grafinis žaidimu variklis buvo pasirinktas kaip pagrindinis:

- PC reikalavimai *Unity* redaktoriui.
- Kadangi *Unity* ir UE4 yra multiplatforminiai grafiniai žaidimų varikiai, HTML5 (WebGL) ir *OpenGL* yra jų sudedamojoje dalyje.
- *Unity* turi didesnį projektų eksportavimo į kitas platformas pasirinkimą nei UE4.
- Patogi redaktoriaus vartotojo sąsaja.

2.5 Papildomi grafiniai varikliai

Papildomi grafiniai varikliai buvo pasirinkti testuojant matematinio modelio pritaikymo galimybes:

- *HTML5* buvo pasirinktas dėl to kad tai yra nauja sistema ir buvo įdomu pamėginti, ar pavyks matematinį modelį pritaikyti jam.
- *OpenGL* buvo pasirinktas dėl to, kad biblioteka skirta realizuoti 2d ir 3d grafikas skirtinguose platformose.
- *Unreal Engine 4* buvo parinktas dėl *Blueprint* vizualios programavimo sistemos.

2.6 Kiti grafiniai varikliai

- UDK (Unreal Engine 3);
- CryEngine 3 SDK;
- Source Engine;
- Leadwerks;
- Torque3D;
- Blender;
- Neoaxis;
- C4 Engine;
- Shive 3D;
- Panda 3D;
- Esenthel Engine;
- iDTech 4;
- Ogre3D (Rendering Engine);
- Irrlicht Engine (Rendering Engine).

2.7 Tyrinėtų virtualių Saulės sistemos modelių analizė

Internete yra nemažai sumodeliuotų virtualių Saulės sistemos modelių, kur yra pateikiama informacija apie Saulės sistemą. Nors yra nemažai galimų variantų, populiarių sistemų kurios yra prieinamos, savybės yra vartotojui yra:

1. Vartotojui yra pateikiama informacija apie planetas ir palydovus.
2. Galimybė sužinoti planetų buvusias, esamas ir būsimas pozicijas keičiant datą.

2.7.1 Solar Sistem Scope ir 3D Solar System Simulator

- Modelį sudaro 3 vaizdavimo būdai (Heliocentrinis, Geometrinis ir panoraminis);
- Tikslios pozicijos visų dangaus kūnų pagal NASA skaičiavimus;
- Scheminiai atstumai ir dydžiai geresniam supratimui apie Planetų paviršius ir judėjimus;
- Galimybė keisti planetų pozicijas jų orbitose;
- Daug įdomių nustatymų kurie leidžia stebėti tam tikrus įvykius ir judėjimus;
- Yra įdiegtas atstumo skaičiuokis kuris matuoja atstumą tarp planetų esant judėjimui;
- Žemės observatorijos nustatymai leidžia stebėti dangaus kūnų įvykius žemės atžvilgiu;

Modelių nuorodas galima rasti [8] ir [9] šaltiniuose.

2.7.2 3D Solar System Web (HTML5)

3D Saulės sistemos aplikacija kuri naudoja HTML5 ir WebGL.

- Modelyje yra naudojamas heliocentrinis ir geometrinis atvadavimas:
- Apytikslės dangaus kūnų pozicijos skirtingu laiku;
- Pateikiama informacija apie planetas;
- Galimybė keisti programoje sistemos struktūra šalinat ar pridedant objektus ir sukurti ekskursiją.

Modelio nuorodą galima rasti [10] šaltinyje.

2.7.3 Solar System Simulator (Unity)

Ši simuliacija yra virtualus Saulės Sistemos žaimas skirtas mokymo tikslams. Šio žaidimo tikslas yra paleisti erdvėlaivius į kosmosą ir vykdyti misijas [11].

- Vartotojas gali suveisti erdvėlaivio duomenis;
- Planetos, mėnulis, erdvėlaiviai veikia pagal realistines elipsines orbitas fazinėje erdvėje;
- Planetų orbitų elementams buvo naudoti NASA planetų duomenų lentelė;
- Planetų orbitos yra apskaičiuotos naudojant 4 eilės Runge-Kutta dalinės išvestinės ir 2 Keplerio dėsnis yra sprendžiamas reikšmėms gauti pagal NASA duomenų lentelę Saulės sistemos planetoms ir mėnuliui.
- Erdvėlaivių trajektorijos yra apskaičiuojamos paleidimo metu naudojant 2 eilės dalinės išvestinės su kraštutinėm sąlygomis. priklausomai nuo vartotojo įvestų duomenų, Žemės pozicijos ir laiko tarpo per kurį erdvėlaivis yra paleistas.

2.7.4 Tyrinėtų Saulės sistemos modelių analizės išvados

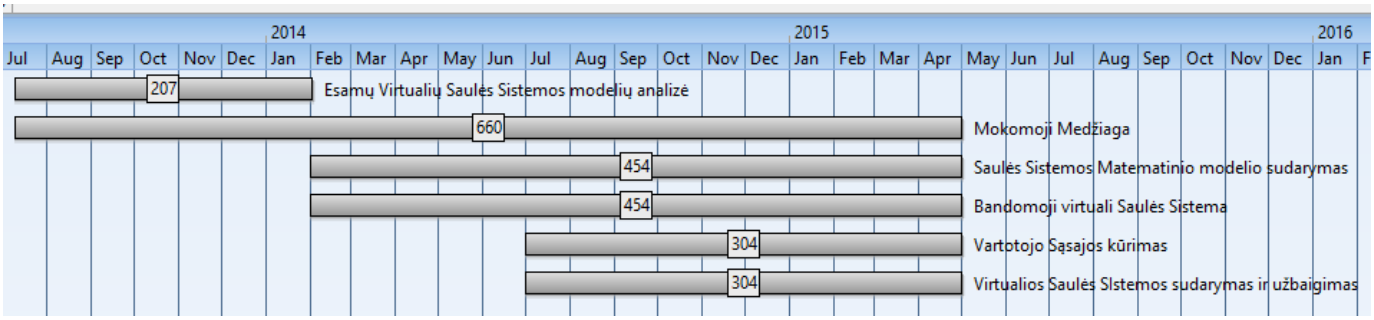
Atlikus esamų virtualių Saulės Sistemos modelių analizę galima teigti, kad didžioji dalis sukurtų virtualių Saulės Sistemos modelių naudoja apskaičiuotas ir nekeičiamas dangaus kūnų orbitas, todėl šie modeliai netiko, išskyrus vartotojo sąsajos pavydžius, mano kuriamai viruliai Saulės sistemai.

2.8 Pradinis projekto aprašymas

Pačioje pradžioje mano darbo tikslas buvo sukurti virtualų Saulės sistemos modelį:

- Kaip mokomąją priemonę:
    - Gauti išsamią informaciją apie Saulės sistemą;
    - Butų paskirstymas pagal mokymo lygius;
    - Naudojamas įgarsinimas ir video medžiaga;
    - Istorinės ir mitologinės žinios apie dangaus kūnus.
- Sudarytas matematinis Saulės sistemos modelis būtų aiškus ir pritaikomas kitiems grafiniams varikliams;
- Planetos ir palydovai, veikiami vienas kitų, judės palei kintamą orbitą.
- Galimybė vartotojui kurti savąją dangaus kūnų sistemą.

2.9 projekto vykdymo planas



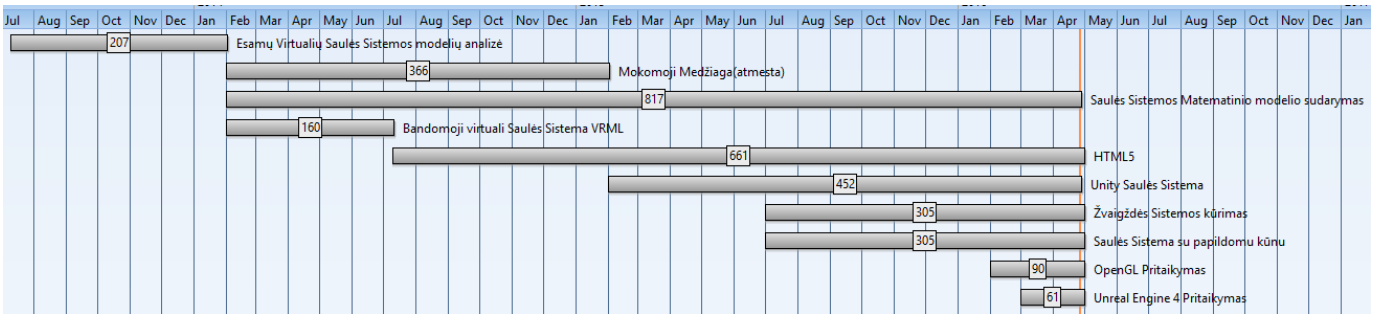
2 pav. Projekto vykdymo planas

3 Darbo eiga ir rezultatai

3.1 Pradiniai objektų parametrai

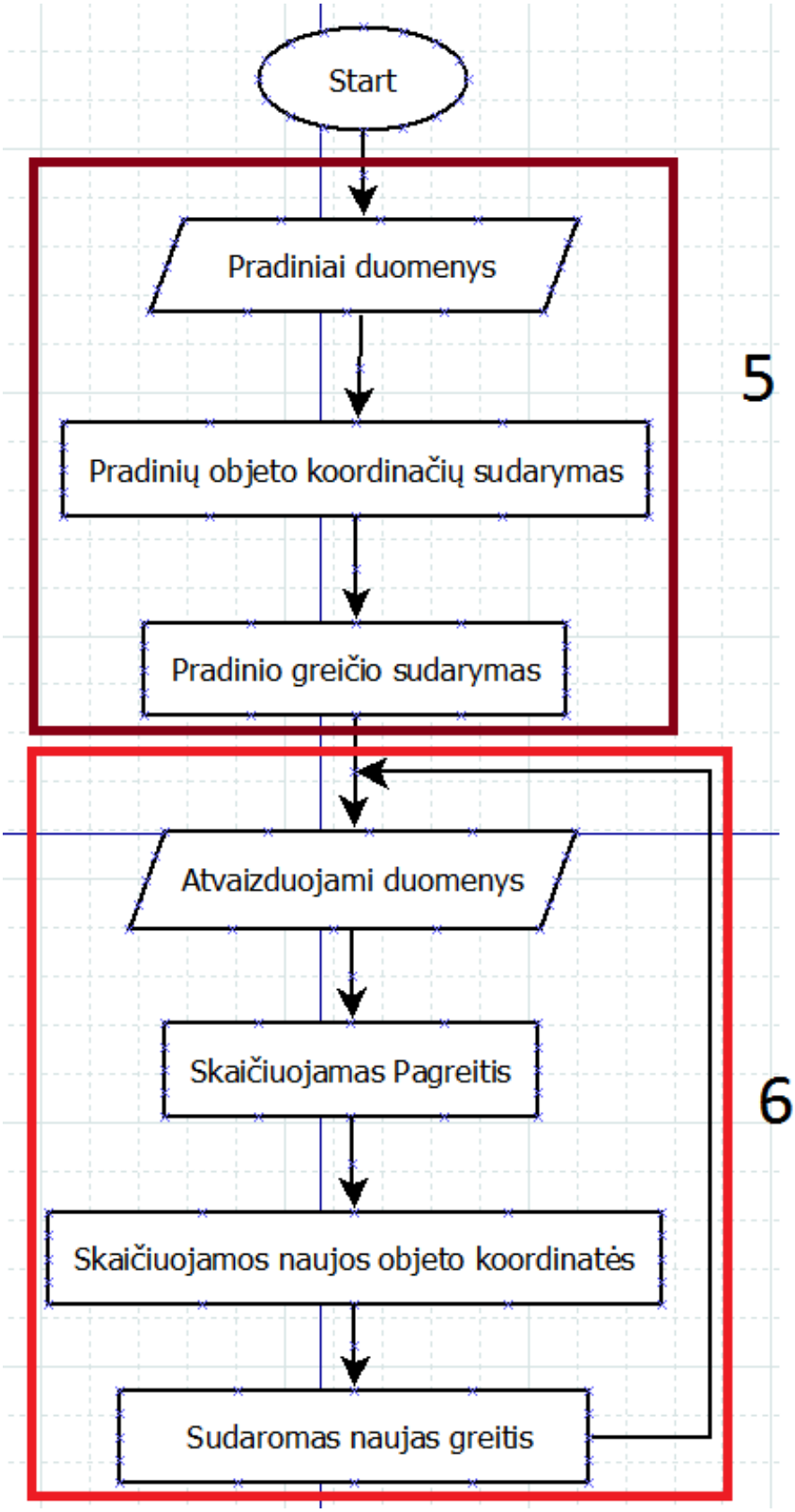
Pradiniams objektų parametrams yra naudojami Nasa parametrai. Jie yra pateikti *Priedas 1* ***Lentelė 3 ir Lentelė 4***.

3.2 Darbo eigos grafas



3. pav. Darbo eigos grafas

3.3 Programos struktūra

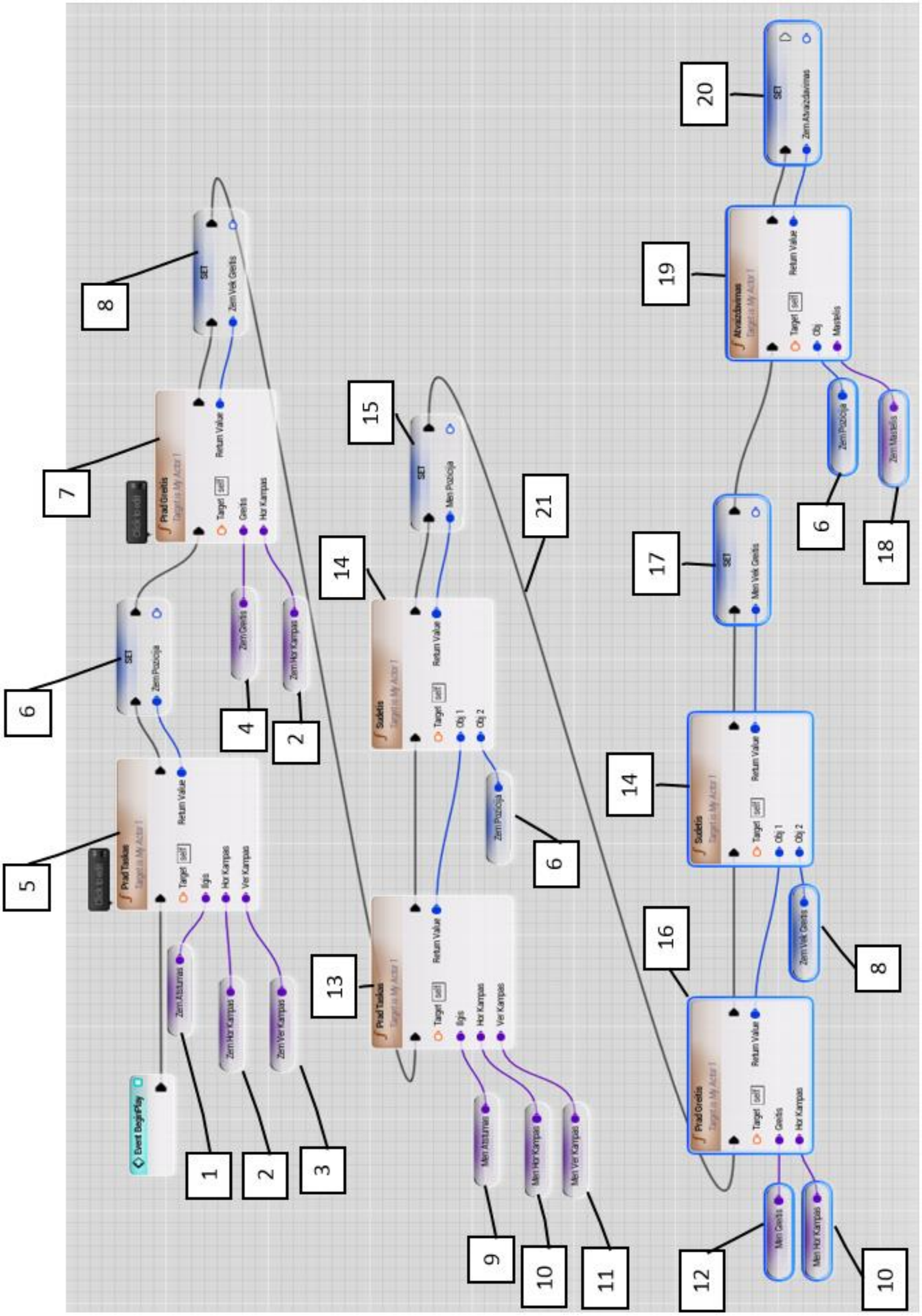


5 pav.

6 pav.

4 pav. Programos struktūra

3.4 Pradinių objektų pozicijų ir greičio vektorinių sudarymo įvykių grafas



5 pav. Pradinių objektų pozicijų ir greičio vektorinių sudarymo įvykių grafas sudarytas naudojant „Blueprint“ sistemą, kuri yra Unreal Engine 4 grafiniame žaidimų variklyje.

1. Planetos atstumas nuo Saulės.

2. Planetos horizontali pozicija laipsniais.

3. Planetos vertikali pozicija laipsniais.

4. Planetos orbitos greitis.

5. ***PradTaskas*** funkcija yra sudaroma naudojant sferinę koordinačių sistemą pagal (7), (8), (9) formules, kur yra įrašomos planetos arba palydovo pradiniai atstumai ir jų horizontalūs, ir vertikalūs kampai nuo koordinačių pradžios. Formules Galima rasti *Priedas 3*.

6. Planetos pozicijos kintamasis.

7. ***PradGreitis*** funkcija apibūdina objekto judėjimo greitį ir kryptį ***PradTaskas*** funkcijoje naudojant (10), (11) formules prieš laikrodžio rodyklę horizontalioje koordinačių plokštumoje arba judėjimo greitį ir kryptį, kur tiesiogiai yra įvestos x, y, z greičio projekcijos koordinates.

8. Planetos greičio projekcijos kintamasis.

9. Palydovo Atstumas nuo Planetos.

10. Palydovo horizontali pozicija laipsniais.

11. Palydovo vertikali pozicija laipsniais.

12 Palydovo orbitos greitis.

13. Naudojant funkcija ***PradTaskas*** sudaromos pradinės palydovo pozicijos koordinatės planetos atžvilgiu.

14. ***Sudėtis*** funkcija apibūdina trimačių vektorių sudėtį. Čia jis yra naudojamas sudaryti supaprastintą 3 kūnų uždavinį, kur prie palydovo pradinės pozicijos ir greičio koordinačių yra pridedamos pradinė pozicija ir greitis.

15. Palydovo pozicijos kintamasis.

16. Naudojant funkcija ***PradGreitis*** sudaromos pradinės palydovo greičio projekcijos koordinatės planetos atžvilgiu.

17. Palydovo greičio projekcijos kintamasis.

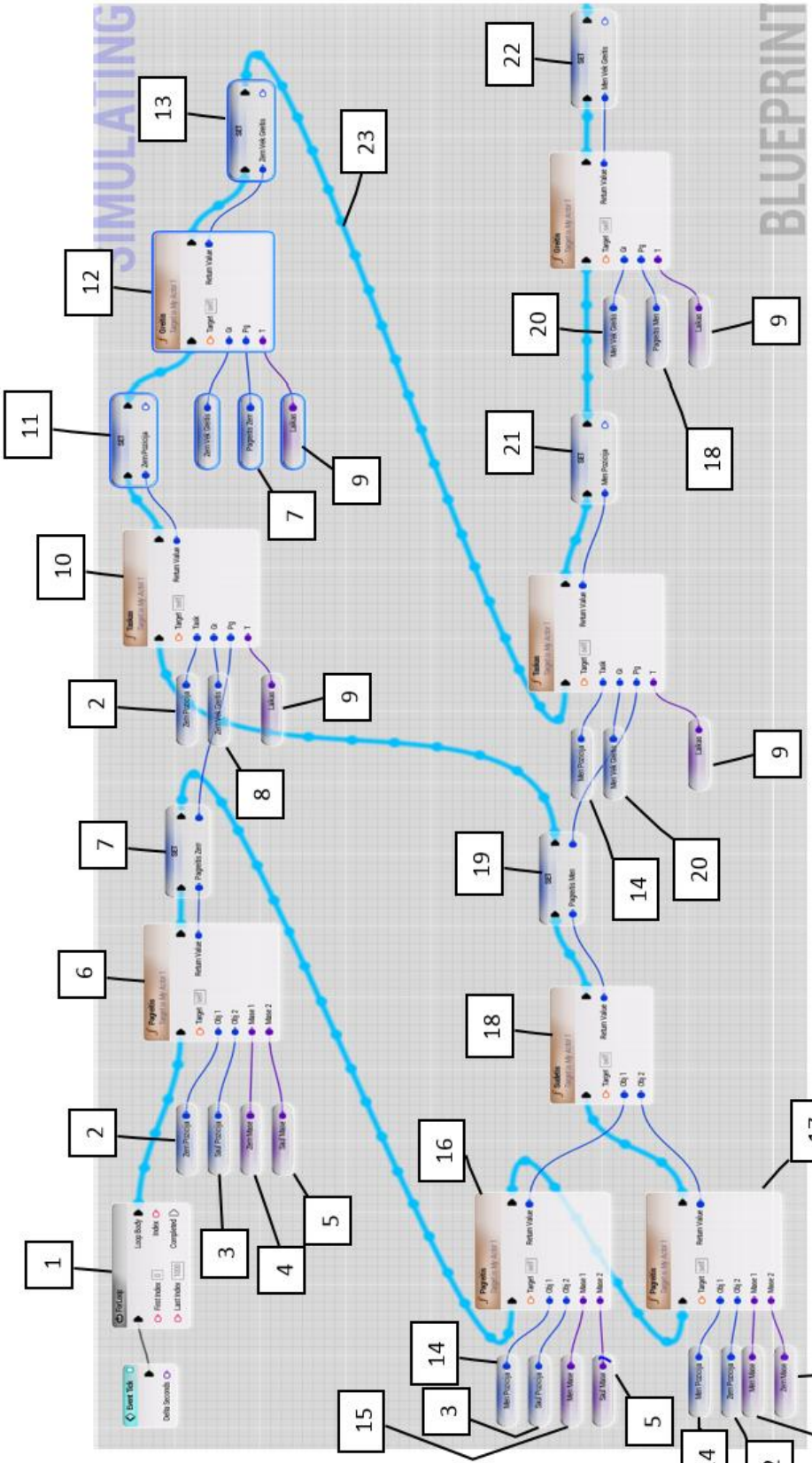
18. Planetos mastelis;

19. ***Atvaizdavimas*** funkcija sudaro planetos atvaizduojamąsias pozicijos koordinates vaizdavimo lauke.

20. Planetos atvaizduojamosios pozicijos koordinatės.

21. Linija parodanti programos atliekamų veiksmų nuoseklumą.

3.5 Objektų judėjimo įvykių grafas



6 pav. Objektų judėjimo įvykių grafas.

1. *for* ciklas.

2. Planetos pozicijos kintamasis;

3. Žvaigždės pozicijos kintamasis;

4. Planetos masė;

5. Žvaigždės masė;

6. ***Pagreitis*** funkcija yra aprašomas naudojant Niutono dviejų kūnų uždavinio formules. Jei nėra naudojamas masės centras tai naudojamos (18), (19), (20) formulės. Jei naudojamas masės centras, tai naudojamos (22), (23), (24) formulės. Formules galima rasti *Priedas 3*.

7. Planetos pagreičio kintamasis.

8. Planetos greičio projekcijos kintamasis.

9. Laiko tarpas.

10. ***Taskas*** funkcija apibūdina nuolat perskaičiuojamą objekto poziciją naudojant tolygiai kintamo judėjimo koordinačių formules (25), (26), (27) praėjus tam tikram laiko tarpui.

11. Naujas planetos pozicijos kintamasis.

12. ***Greitis*** funkcija apibūdina tolygiai kintamą greitį, pagal (28), (29), (30) formules. Formules galima rasti *Priedas 3*.

13. Naujas planetos greičio projekcijos kintamasis.

14. Palydovo pozicijos kintamasis.

15. Palydovo masė.

16. Palydovo pagreitis žvaigždės atžvilgiu.

17. Palydovo pagreitis planetos atžvilgiu.

18. Susumuojamas pagreitis.

19. Palydovo pagreičio kintamasis.

20. Palydovo greičio projekcijos kintamasis.

21. Naujas palydovo pozicijos kintamasis

22. Naujas palydovo greičio projekcijos kintamasis.

23. Linija parodanti programos atliekamų veiksmų nuoseklumą.

3.6 HTML5 Saulės sistema

HTML5 Saulės sistema buvo sudaryta pasinaudojus *3D Solar System on WebGL* [12], kur buvo jau apipavidalintas trimatis objekto atvaizdavimas ir kameros valdymas.

Programą sudaro 4 pagrindiniai elementai, ***solar.html***, ***solar.js*** – trimatis objektų apipavidalinimas ir pozicijų skaičiavimo skribtas, ***RollControls.js*** – kameros valdymo skribtas, ***three.min.js*** – Javascript 3D biblioteka [4].

3.6.1 Funkcijų sudarymas

Pirmiausia yra surandama struktūra apipavidalinantį trimatį vektorių ***three.min.js*** bibliotekoje. Struktūra yra papildoma funkcijomis sudarytomis Saulės sistemos matematinio modelio formulių, kurios yra gražinamos kaip vektorius. Matematinio modelio formules galite rasti *Priedas 3***.**

Pradinių objekto koordinačių sudarymas pagal (7), (8), (9) formules:

```
VecPozicija: function(a, b, c){
    var rad = b*Math.PI/180;
    var rad2 = c*Math.PI/180;
    this.x = Math.cos(rad2)*Math.cos(rad)*a;
    this.z = Math.cos(rad2)*Math.sin(rad)*a;
    this.y = Math.sin(rad2)*a;
    return this
},
```

7 pav. Pradinė objektų pozicijos funkcija. Ši funkcija atitinka (*žr. 5 pav.*) *PradTaskas* funkciją.

Čia

a – objekto atstumas;

b – kampas tarp x, z plokštumų;

c – kampas tarp z, y plokštumos;

Jei objektas yra palydovas, prie jo pozicijos koordinačių yra pridedamos planetos pozicijos koordinatės. Tokiu būdu yra sudaromas trijų kūnų uždavinys, kur atvaizduojama palydovo pozicija yra vaizduojama Saulės atžvilgiu. Šiam veiksmui atlikti (*žr. 5 pav.*) yra naudoja funkcija ***Sudetis***, kur yra apipavidalintas vektoriaus koordinačių sudėtis.

Panašiai yra sudaroma ir tolygiai kintamo greičio funkcija pagal (10), (11) formules, čia jis pavadintas ***VekPozGreitis***. Jei sudaromas palydovo greitis, tai prie jo pridedamas planetos greitis, kai sudaromas trijų kūnų uždavinys.

Programoje nėra naudojamas masės centras, todėl pagreičio funkcijai apipavidalinti naudojamos (18), (19), (20) formulės.

```
VecPagreitis: function(a, b, Mas1, Mas2){
    var sd = (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y) + (a.z-b.z)*(a.z-b.z);
    var fg = -G*(Mas1 + Mas2)/Math.sqrt(sd);
    fg = fg/sd;
    this.x = fg*(a.x-b.x);
    this.y = fg*(a.y-b.y);
    this.z = fg*(a.z-b.z);
    return this
},
```

8 pav. Pagreičio funkcija be masės centro. Ši funkcija atitinka (*žr. 6 pav.*) *Pagreitis* funkciją.

Kadangi masės centras ir ciklas cikle nėra naudojamas, buvo iškilusios problema. Jei spręsti buvo sudarytos laikinosios funkcijos pagrečiams sumuoti ***VeckPagSuma***, kur sudedami planetos be palydovo pagreičiai ir ***VeckPagSuma2***, jei planeta, šiuo atveju žemė, turi palydovą. Čia planetai neprisumuojamas pats palydovo pagreitis, nes pagal supaprastintą trijų kūnų uždavinį, trečio kūno masė yra labai maža ir jis neturi įtakos kitų dviejų kūnų judėjimui apie bendrą jų masės centrą.

Naujos objekto pozicija yra skaičiuojama naudojant funkcija ***VecTaskas,*** kuri sudaryta pagal (25), (26), (27) tolygiai kintamo judėjimo koordinačių formules. Čia perskaičiuojama objekto pozicija praėjus tam tikram laiko tarpui,

Tolygiai kintamo greičio funkcija ***VecGreitis***, yra sudarytas pagal (28), (29), (30) formules. Čia perskaičiuojamas objekto greitis praėjus tam tikram laiko tarpui. Formules galima rasti *Priedas 3*.

Kai jau yra gaunamos kintamos objektų koordinatės yra sudaromos funkcijos pagal (1), (2), (3) planetų koordinatėm vaizduoti ir pagal (6), (7), (8) palydovo koordinačių atvaizdavimas (*žr. 3.11.1 poskyrį*):

```
VecAtvaizdavimas(a, b){
    this.x = a.x/b;
    this.y = a.y/b;
    this.z = -(a.z/b);
    return this
},
```

9 pav. Funkcija planetos atvaizdavimo koordinatėms sudaryti.

Ir palydovo atvaizdavimo taškas (4), (5), (6):

```
VecPalydAtvaizdavimas: function(a, b, c, d){
    this.x = (a.x - b.x)/c + d.x;
    this.y = (a.y - b.y)/c + d.y;
    this.z = (-(a.z - b.z)/c) + d.z;
    return this
},
```

10 pav. Funkcija palydovo atvaizdavimo koordinatėms sudaryti.

Vektoriaus koordinatė *z* turi minusą, nes *HTML5* teigiama objekto judėjimo kryptis juda palei laikrodžio rodyklę.

3.7 *OpenGL* ir *Unity* bendros funkcijos

Kad matematinis modelis veikų teisingai reikėjo sudaryti ***double*** tikslumo vektorių apibūdinančią struktūros *x, y, z* koordinates. Kadangi *OpenGL* ir *Unity* projektai naudoja C# kodą, jiems naudojama ta pati vektorių apipavidalinanti struktūra:

```
public struct Position2
{
    public double x;
    public double y;
    public double z;

    public Position2(double x, double y, double z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }
```

11 pav. Struktūra apipavidalinanti vektorių *double* tikslumu.

OpenGL vektoriaus struktūros aprašas yra ***SolarSystem.cs***, *Unity* projekte struktūra yra aprašyta ***Assets/Bendro Naudojimo/Judėjimas.cs***. Vienas iš skirtumu tarp *Unity* ir *OpenGL* C# projektų yra matematinio paketo iškvietimas, *Unity* naudoja ***Mathf***, kur reikšmes skaičiuoja ***float*** tikslumu, o *OpenGL* naudoja ***Math***, kur reikšmės skaičiuojamos ***double*** tikslumu.

Kaip ir (*žr. 7 pav.*) funkcijoje, naudojant matematinį Saulės sistemos modelio (7), (8), (9) formules sudaromos pradinės objekto koordinatės. Formulės yra aprašytos *Priedas 3*.

```
public Position2 PradTaskas(double ilgis, double HorKampas, double VerKampas)
{
    double rad = HorKampas * Mathf.PI / 180;
    double rad2 = VerKampas * Mathf.PI / 180;
    double x = Mathf.Cos((float)rad2) * Mathf.Cos((float)rad) * ilgis;
    double z = Mathf.Cos((float)rad2) * Mathf.Sin((float)rad) * ilgis;
    double y = Mathf.Sin((float)rad2) * ilgis;
    return new Position2(x, y, z);
}
```

12 pav. Pradinių objektų koordinačių funkcija.

Masės Centas aprašytas pagal (12), (13), (14) formules yra išskaidomas dalimis ir panaudojami pozicijos ir greičio sudarymo cikle. Cikle sudaryta objekto masė yra sumuojama į vieną bendrą visų objektų masę ir tuo pačiu naujai sudaryta objekto pozicijos koordinatės yra sudauginama su jos mase, ir sumuojama į bendra kintamąjį.

Pasibaigus ciklui, visų objektų masių ir koordinačių sumos kintamasis yra padalijamas iš visų objektų masių sumos, kurios rezultatas yra masės centro koordinatės.

Pagal (15), (16), (17) formules iš esamų pradinių koordinačių yra atimamas masės centras ir sudaromos naujos pradinės objektų koordinatės.

Kadangi masių centras yra naudojamas, yra sudaroma funkcija ***PagreitisC*** pagal (16), (17), (18) formules.

Naudojant (25), (26), (27), (28), (29), (30) formules yra sudaromos objekto pozicijos ir greičio funkcijos ir kaip ir (*žr. 9 pav.*) ir (*žr. 10 pav*.), pagal (1), (2), (3), (4), (5), (6) formules yra sudaromos objektų pozicijos atvaizdavimo funkcijos.

3.8 *OpenGL* Saulės sistema

OpenGL sistemai sudaryti pasinaudojau Vasily Tserekh demonstraciniu virtualiu saulės sistemos modeliu [13]. Jame autorius apibūdina, vaizduojamo lango sudarymą naudojant ***Windows Forms***, trimatį objekto atvaizdavimą ir kameros valdymą. Projektas naudoja ***Tao.OpenGL.dll*** Framework ir autoriaus sukurtą ***Shadowengine.dll***..

Objekto sferai apipavidalinti yra sudaryta *Create()* operatorius *Planet.cs* skripte, čia yra sudaromas sferos dydis ir galimybė naudoti tekstūrą.

Objektų judėjimui yra naudojamas *Paint()* operatorius *Planet.cs* skripte, čia yra įvedamos sukurto objekto atvaizduojamosios koordinatės ir kiekvieną kartą į ją kreipiantis yra sudaroma nauja objekto pozicija.

3.9 *Unity* Saulės sistema

Objektą Unity grafinio variklio redaktoriuje galima sukurti dviem būdais:

1. Pirmas būdas yra naudojat pačiame redaktoriuje jau apipavidalintas geometrines dvimates, trimates formas kaip atvaizdavimo objektus arba sudarytas formas iš kitų modeliavimo įrankių ir jiems yra priskiriami atskiri skriptai kurie pasiima atvaizduojamąsias pozicijos koordinates iš pagrindinio skripto kur jie yra skaičiuojami.
2. Antras būdas yra skripte, kur atliekami kreipiniai ir skaičiavimai, galima pagrindines geometrines figūras, kurios yra redaktoriuje, apipavidalinti kodu ir priskirti jiems atvaizdavimo koordinates. Pavyzdį galima rasti ***Assets/Set1/Planeta.cs*** skripte.

3.10 *Unreal Engine 4* Saulės sistema

Unreal Engine 4 yra paskutinis grafinis variklis kuriam pritaikiau Saulės sistemos matematinį modelį. Buvo bandymas daryti projektą *Blueprint* programavimo sistema, bet kadangi visos reikšmės buvo gražinamos ***float*** tikslumu, skripte kaip ir kituose mano projektuose struktūra apipavidalinančia *x, y, z* vektorių ***double*** tikslumu. Vienas esminis skirtumas UR4 koordinačių atvaizdavime, z plokštuma atvaizduoja aukštį vietoj y plokštumos. Apipavidalintas funkcijas galima rasti *Objetai2.cpp* skripte.

```
USTRUCT()
struct FPosition2
{
    GENERATED_USTRUCT_BODY()

    double x;
    double y;
    double z;
```

13 pav. Struktūra apibūdinanti vektoriaus koordinates.

```
FPosition2 FPosition2::PradTaskas(double ilgis, double HorKampas, double VerKampas)
{
    double rad = HorKampas * PI / 180;
    double rad2 = VerKampas * PI / 180;
    double x = cos(rad2) * cos(rad) * ilgis;
    double y = cos(rad2) * sin(rad) * ilgis;
    double z = sin(rad2) * ilgis;
    return FPosition2{ x, y, z };
}
```

25 pav. Pradinė objekto pozicija.

Redaktoriuje skriptui yra suteikiamas šablono (*Blueprint*) statusas, kur atvaizduojamosios koordinatės yra pasiimamos ir priskiriamos sukurtiems objektams.

3.11 Iškilusios problemos ir jų sprendimas

3.11.1 Atvaizdavimas

Kadangi projekte naudojami realūs dangaus kūnų atstumai ir yra viršijamos grafinio variklio atvaizdavimo ribos, todėl yra naudojamas mastelis, kuris sumažina vizualiai objektų pozicijas į leistinas atvaizdavimo ribas.

Planetos atvaizdavimas Saulės atžvilgiu:

$$x_p = \frac{x_{planeta}}{mastelis} \quad (1)$$

$$y_p = \frac{y_{planeta}}{mastelis} \quad (2)$$

$$x_p = \frac{z_{planeta}}{mastelis} \quad (3)$$

Čia

x_p, y_p, z_p – planetos atvaizduojamosios koordinatės;

$x_{planeta}, y_{planeta}, z_{planeta}$ – planetos pozicijos koordinatės;

Kadangi palydovas Saulės atžvilgiu yra mažas kūnas kuris yra veikiamas planetos, mažinat mastelį, mažėja ir vaizduojamasis palydovo atstumas, todėl iškyla problema kaip jį atvaizduoti.

Problemos spendimas yra iš palydovo pozicijos koordinačių atimti planetos pozicijos koordinates ir gautą skirtumą padalinti iš palydovo mastelio atstumu nuo žemės ir gautam rezultatui pridėti atvaizduojamąsias žemės koordinates:

$$x_{pal} = \frac{x_{palydovas} - x_{planeta}}{PalMastelis} + x_p; \quad (4)$$

$$y_{pal} = \frac{y_{palydovas} - y_{planeta}}{PalMastelis} + y_p; \quad (5)$$

$$z_{pal} = \frac{z_{palydovas} - z_{planeta}}{PalMastelis} + z_p; \quad (6)$$

Čia

$x_{pal}, y_{pal}, z_{pal}$ – palydovo atvaizduojamosios koordinatės;

$x_{palydovas}, y_{palydovas}, z_{palydovas}$ – palydovo pozicijos koordinatės;

$PalMastelis$ – palydovo mastelis žemės atžvilgiu.

Jei palydovas per nelyk nutolsta nuo planetos, palydovas atvaizduojamas kaip planeta Saulės atžvilgiu.

3.11.2 HTML5 Saulės sistemos iškilusios problemos ir jų sprendimas

HTML5 buvo bandyta padaryti tokiu pat principu kaip ir kituose sudarytose sistemose kaip *Unity* C# ar UR4 C++, bet dėl veiksmų apkrovos kuris susidaro naudojas ciklą cikle programa sustoja nespėdama apdoroti informacijos ir gaudama didelę apkrovą pastringa. Pagal oficialia informacija [14] *HTML5* sunkiai susidoroja su kompleksinėmis programos struktūromis. Todėl projekte naudojamas nuoseklus objektų veiksmų atlikimas ir vienas ciklas *for* pozicijų skaičiavimui ir atvaizdavimui pagreitinti.

Kadangi masių centras nebuvo naudojamas ir naudojamas nuoseklus objektų duomenų veiksmų atlikimas be ciklo sumavimo, buvo sudaromos laikinosios funkcijos pagreičiui susumuoti, kaip *VeckPagSuma* ir *VeckPagSuma2*. Funkcijos apipavidalintos ***Three.min.js*** bibliotekoje.

3.11.3 *Unreal Engine 4* Saulė sistemos iškilusios problemos ir jų sprendimas

Buvo bandyta sudaryti Saulės sistemą naudojant *Blueprint* vizualia skriptinimo sistema ir buvo sudarytas 3 kūnų uždavinys (*5 pav.* ir *6 pav.*), bet kadangi funkcijos gražina reikšmes ***float*** tikslumu, atsiranda skaičiavimo netikslumų, kai naudojama gravitacijos konstanta.

Todėl teko sukonstruoti C++ skripte struktūrą kuri apipavidalintų trimatį vektorių (žr. *13 pav.*) ***double*** tikslumu. Gautos pozicijos atvaizduojamos reikšmės priskiriamos variklio pakete esančiam vektoriui, kuris perduoda reikšmes grafinio variklio redaktoriui. Naudojant *Blueprint* sistemą kintamos reikšmės yra pasiimamos ir naudojamos objektų pozicijai atvaizduoti.

3.11.4 OpenGL Saulės sistemos iškilusios problemos ir jų sprendimas

OpenGL savo pakete neturėjo apipavidalinto vektoriaus, todėl jam sudaroma vektoriaus struktūra, kur apipavidalinamos vektoriaus koordinatės ***double*** tikslumu. Naudojam autoriaus sukonstruotą sferos apipavidalinimą yra sudarytas reikalingas objektų kiekis ir jų judėjimas atvaizduotas naudojant *Paint()* operatorių, kur priskiriamos atvaizduojamos koordinatės.

3.11.5 *Unity* Saulės sistemos iškilusios problemos ir jų sprendimas

Kadangi variklio pakete esantis vektorius gražina tik ***float*** tikslumu, teko sudaryto naują vektoriaus struktūrą (pav), kur apipavidalinamos vektoriaus koordinatės ***double*** tikslumu.

Kad parašytas kodas veiktų, išskyrus struktūros aprašymą, skriptas kur tas kodas yra būtinai turi būti priskirtas objektui esančiam atvadavimo lauke. Skriptas kur aprašomos funkcijos ir operatoriai rekomenduojama priskirti prie nematomų tuščių objektų, kurie yra žaidimo variklio redaktoriuje.

Žvaigždės sistemos kūrime viena iš problemų buvo pačio sukurto objekto sunaikinimas. Kiekvieną kartą kai buvo paleidžiamas skriptas, kur suvesti duomenys yra panaudojami objektų sukūrimui ir jų judėjimui atvaizduoti naudojant *Unity* operatorių *OnEnable()*, ne visi objektai būdavo sunaikinami grįžtant į meniu. Kai operatorius būdavo aktyvuojamas vėl, nauji objektai susidubliuodavo su senais sukeldamas atvaizdavimo kokybės ir greičio problemas jei pasirinktas objektų kiekis buvo didelis. Problema buvo išspęsta pasitelkus *foreach* ciklą, kuris suranda pasiritame masyve visus objektų esančius elementus ir juos ištrina.

3.12 Projektų palyginimas ir analizė

3.12.1 Atvaizdavimo sparta

Greičiausiai su matematiniu modeliu susidoroja HTML5, nes jam nereikia įdiegti papildomų atvadavimo įrankių kaip Adobe flash, kuris ne visais atvejais turi prieigą prie GPU (*Graphics Processing Unit*) ir naudoja CPU (*Central Processing Unit*) dėl kurio programa gali labai sulėtėti. *HTML5* ištaiso šia problemą naudodamas bendrą sąsają ir skaičiuojami ir atvaizduojami elementai susibendrauja greičiau.

Lėčiausiai su matematiniu modeliu susidorojo *OpenGL* Saulės sistemos projektas, nes pats *OpenGL* nėra grafinis variklis, su juo sukurta projekto sparta labai priklauso nuo kompiuterio specifikacijų ar kodo aprašo, naudojamų įrankių, ir pačios bibliotekos versijos ir kaip ji išnaudoja esamus kompiuterio resursus.

Iš dabartinių pasirinktų grafinių variklių Saulės sistemos matematinį modelį atvaizduoja *Unreal Engine 4*. Nors modelio atvaizdavimas nėra toks greitas kaip *HTML5*, jo pasirenkama optimizacija leidžia nustatinėti kompiuterio resursus tarp vaizdo kokybės, kuo *UE4* yra gerai žinomas, ir atvaizdavimo greičio.

Nors *Unity* susitvarko lėčiau nei *UE4* su matematiniu modeliu, nes pats projekto skaičiavimo resursų paskirstymas nėra dar pilnai optimizuotas. Jis pakankamai gerai susitvarko matematiniu modeliu ir dėl redaktoriaus patogios vartotojo sąsajos, galimybės projektą pritaikyti daugelyje platformų, jis buvo pasirinktas kaip pagrindinis grafinis varikis tolimesniam projektų plėtojimui.

3.12.2 Projekto sudarymo sudėtingumas

Sudėtingiausia projektą buvo sudaryti naudojant *HTML5* Javascript, nes nebuvo klaidų sąrašo (error list) galimybės, kur parodomos sintaksės klaidos dėl kurio rašomas algoritmas nesuveikia.

Lengviausias projekto sudarymas buvo *Unreal Engine 4* grafiniu varikliu, nors dėl skaičiavimų tikslumų buvo naudojamas C++ kodo skiptinimo variantas objektų pozicijoms skaičiuoti, atvaizdavimui buvo naudojama *Blueprint* sistema, kur suskaičiuotos reikšmės buvo priskiriamos objektams ir sudaroma vartotojo sąsaja.

Unity ir *OpenGL* projektų sudarymas tai pat nebuvo sudėtingas naudojant C# kalbą, tik *OpenGL* projekte buvo sudėtingiau išsiaiškinti kai reikia apipavidalinti trimatę geometrinę figūrą ir ja judinti.

3.12.3 Projektų testavimas

Matematinis modelis buvo pritaikomas grafiniams varikliams keliais būdais ir buvo tikrinama kuris iš jų tinka geriausiai.

HTML5 Saulės sistemos projekto aplankale yra bandomieji projektai pavadinimu *bandymas1* ir *bandymas2*.

HTML5 ***Bandymas1*** – sudarytos funkcijos skaičiuoja ***x, y, z*** koordinates atskirai. *HTML5* ***Bandymas2*** – objekto pozicijos skaičiavimas sudarant funkcijas ***Three.min.js*** bibliotekoje, kur trimačio vektoriaus eilutė yra apipavidalinta. Patikrinus atliekamą skaičiavimo apkrovą, buvo nuspręsta plėtoti ***bandymas2*** variantą, nes čia yra sumažinamas reikalingas skaičiavimų kiekis tris kartus, kurios rezultatas pagreitina atvaizdavimo greitį ir leidžia naudoti mažesnį laiko tarpo įvedimą tikslesnėms objektų orbitoms skaičiuoti.

HTML5 projekte tai pat buvo bandymas naudoti daugialypių masyvų skaičiavimą, kurių pagalba sukčiavimo aprašas būtų sutrumpintas. Bet dėl HTML5 trukumų buvo iškilusios problemos (*žr. 3,11.2 poskyris*), todėl šis variantas, kuris buvo pritaikytas kitiems parinktiems grafiniams varikliams, buvo atmestas.

Unity Saulės sistemos projekte ***Assets/bandymai*** buvo sudaromos bandomosios sistemos. Čia ***Bandymas1*** buvo vienas iš pirmųjų sėkmingai pritaikytas matematinis modelis. Jame yra naudojamas integruotas trimačio vektoriaus apipavidalinimas *Vector3()*. Pačios funkcijos yra apipavidalintos *Judijimas.cs* skripte. Čia kiekvienam atvaizduojamajam objektui buvo priskirti skriptai, kur buvo atliekami jo orbitos judėjimo skaičiavimai. Šio metodo teko atsisakyti, nes projektas veikė lėtai. Priežastis buvo, kad 22 objektu skriptai atlikdavo skaičiavimus tuo pačiu metu, apsunkindami proceso darbą.

Bandymo keliu objektų skaičiavimas buvo sutrauktas į vieną skriptą naudojant daugialypius masyvus. Šis metodas buvo pritaikytas ***Set1***, ***Set2***, ***Set3*** projekto dalims.

Unity ***Bandymas2*** ir ***Bandymas3*** buvo sudaryti testuojant projekto veikimą su ir be masės centro, ir tuo pačiu buvo tikrinamas projektų veikimas sudarant ***Bandymas2*** projekte naudojant ciklus, ***Bandymas3*** **–** metodika taikyta HTM5 Saulės sistemos projektui. Gauti rezultatai parodė kad ***Bandymas2*** ciklų naudojimas skaičiuojant objektų pozicijas neapkrauna veiklos ir ***Bandymas3*** naudojamas masės centras patikslina objektų orbitas, kur tai pat yra įtraukiamas planetų ir įvesto svetimo kūno poveikis Saulei. Šie projektai buvo panaudoti sudarant ***Set3***, čia yra aprašyta „*Saulės sistema su svetimo kūno įvestimi*“, kuri yra *Unity* Saulės sistemos viena iš pagrindinių projekto dalių.

OpenGl Saulės sistemai buvo pritaikytas *Unity* ***Set3*** arba „*Saulės sistema su svetimo kūno įvestimi*“ variantas. Vienintelis esminis skirtumas tarp projektų skaičiavimų yra tik objektų sudarymas ir

atvaizduojamų koordinačių priskirimas trimačiams objektams. Funkcijų ir kreipinių aprašas yra ***SolarSystem.cs*** skipte.

Unreal Engine 4 Saulės sistemos projektui buvo tai pat pritaikytas *Unity* **Set3** arba „*Saulės sistema su svetimo kūno įvestimi"* variantas.

Unreal Engine 4 ***Bandymas1*** projektas yra bandymas sudaryti Saulės sistemos modelį naudojant „Blueprint" skiptinimo sistemą. Čia grafinio variklio redaktoriuje ***MyMyActor1*** šablono klasėje aprašytas pats „Blueprint" skriptinimas, jis yra naudojamas *5 pav.* ir *6 pav.*, o ***MyActor1.h*** ir ***MyActor1.cpp*** apipavidalintos funkcijos kurios yra iškviečiamos į šablono klasę.

3.12.4 Pradinių duomenų parinkimas projektui

Tai pat *Unity* ***Bandymas1*** projekte buvo bandomi objektų pradinių duomenų veikimo testavimas. Čia buvo nustatyta, kad projekto veikimui tinka planetos ar palydovo tolimiausias atstumas ir mažiausias orbitos greitis arba vidutinis atstumas ir vidutinis orbitos greitis. Kitu atveju naudojant artimiausią atstumą ir didžiausią orbitos greitį kaip pradinius objekto duomenis, objektai nespėja sulėtėti ir sudaro didžiules elipses kurios kertasi su kitų objektų orbitomis. Naudojamus duomenis galima rasti *Priedas 1* ***Lentelė 3 ir Lentelė 4.***

3.12.5 Projektų mastelių ribos

Kadangi projekte yra naudojami realūs dangaus kūnų atstumai, kurie viršija naudojamų grafinių variklių atvaizdavimo galimybes. Problemai spręsti buvo pasitelktas mastelis (*žr. 3.11.1 poskyrį*).

Priklausomai nuo atvaizduojamų objektų dydžių ir grafinio variklio galimybių, projektams buvo suteikti skirtingi masteliai.

HTML5 projekte buvo Merkurijui, Venerai, Žemei, Marsui atstumas sumažintas 5000000 kartų, Jupiteriui – 9000000, Saturnui – 10000000, Uranui ir Neptūnui – 12000000, Plutonui – 14000000 Menuliui atstumas nuo Žemės buvo naudojamas 120000, čia mastelis buvo naudojamas palydovo atvaizdavimui toliau nuo planetos, nes palydovo atstumas Saulės atžvilgiu yra labai mažas ir mažinant planetos atstumą, kuriam šis palydovas priklauso, palydovas tampa nematomu. Priežastis kodėl planetoms buvo suteikiamas skirtingas mastelis yra dėl vaizdavimo patogumo.

OpenGL projekte kaip ir *HTML5* projekte, planetoms buvo suteikiamas skirtingas mastelis vaizdavimo patogumui, naudojamus dydžius galima rasti ***SolarSystem.cs*** Mastelis masyve.

Unity projekte, kuris yra suskirstytas į 3 dalis, mastelis informacinėje dalyje ***Assets/Set2*** yra parašytas **PlanetosIrPalydovai.cs**, čia tai pat naudojami skirtingi masteliai patogesniam atvaizdavimui. ***Assets/Set1*** *Žvaigždės kūrimo sistemoje* yra naudojamas mastelio dydis 60000000, o palydovui 1600000. Mastelis yra **Planeta.cs** skripte. Sudarant planetas ir palydovus objektų atstumas turi viršyti mastelį, kitu atveju, priklausomai kokį vartotojas pasirinko objekto dydį, planeta ar palydovas nesimatys. ***Assets/Set3***, kur sudaryta Saulės sistema su svetimo kūno įvestimi, yra naudojamas vienodas mastelio dydis planetoms – 60000000, kuris yra aprašyras **Kunai.cs** skripte. Taip buvo pasirinka dėl tikslesnio svetimo kūno poveikio sistemai.

Unreal Engine 4 projekto mastelio dydis visoms planetoms buvo taikomas vienodas. Jis yra randamas ***Source/SauesSistemaUE4CPlus/private/objektai2.cpp*** skirpte.

Projektuose, išskyrus *Unity* **Set2**, palydovų atvaizdavimui yra sudaryta papildoma sąlyga. Jei palydovas nutolsta, veikiamas kitų kūnų, nuo planetos dvigubai didesniu atstumu, palydovui yra suteikiamas planetos atvaizdavimas.

3.12.6 Objektų kiekis

Visuose projektuose yra naudojamas ribotas objektų kiekis. Pasirinktas kiekis turi didžiulę įtaką pačių objektų judėjimo atvaizdavimo greičiui.

HTML5, OpenGL, Unreal Engine 4 ir *Unity* ***Set3*** svetimo kūno įvesties projektuose yra naudojama 11 objektų: 9 planetos, 1 palydovas ir svetimo kūno įvestis.

Unity **Set2** yra atvaizduoti 22 objektai: 9 planetos ir 13 palydovų.

Unity **Set1** *Žvaigždės Sistemos Kūrimas* leidžia vartotojui sudaryti 1 pagrindinę žvaigždę, 20 planetų ir jiems priskirti iki 4 palydovų. Padidinus masyvų kiekį skriptuose, sudaromų objektų kiekį galima padindinti, bet didelis objektų kiekis viršijantis nustatytą, labai gali sulėtinti programos veikimą. Viskas priklauso nuo naudojamos įrangos kurioje programa yra naudojama.

3.13 Gauti rezultatai

HTML5 Saulės sistema leidžia vartotojui stebėti realistinę Saulės sistemą, keisti jos atvaizdavimo greitį ir įvesti svetimo kūno parametrus ir stebėti jo poveikį sistemai. Kadangi masių centras yra nenaudojamas, kūnas neveikia Saulės kuris yra nejudamas objektas.

OpenGL Saulės Sistema kaip ir HTML5 Saulės sistema vartotojui leidžia stebėti realistinį Saulės sistemos judėjimą, keisti jos atvaizdavimo greitį ir įvesti svetimo kūno parametrus ir stebėti jo poveikį sistemai. Įvestas kūnas veikia ir yra veikiamas visų kūnų kurie yra atvaizduojami sistemoje.

Unreal Engine 4 Saulės sistema yra tokia pat kaip ir *OpenGL* Saulės sistema, kur vartotojui leidžia stebėti realistinį Saulės sistemos judėjimą, keisti jos atvaizdavimo greitį ir įvesti svetimo kūno parametrus ir stebėti jo poveikį sistemai. Įvestas kūnas veikia ir yra veikiamas visų kūnų kurie yra atvaizduojami sistemoje. Šio projekto planetų mastelis yra vienodas.

Unity Saulės sistema yra suskirstyta į tris projektus:

3. Žvaigždės sistemos kūrimas, kur vartotojas gali sukurti savo žvaigždės sistemą, šioje sistemoje vartotojas gali eksperimentuoti su orbitos judėjimu ieškodamas planetos atitinkamo atstumo, greičio ir masės priklausančios nuo pasirinktos Žvaigždės masės. Sukurti dvinarę žvaigždžių ar planetų sistemą. Vartotojas tai pat gali pasirinkti vieną iš 4 dangaus fonų (skybox), įjungti ir iš jungti muziką. Yra galimybė sukurtą sistemą išsaugoti.
4. Informacinė dalis. Vartotojas gali stebėti realistinę saulės sistemą ir sustabdžius simuliacija paspausti ant planetos ir atsiranda meniu aprašančią minimalią informaciją apie ją, tai pat yra galimybė keisti atvaizdavimo greitį ir metų įvestį, kur atvaduojamos apytikslės planetų pozicijos kurie priklauso nuo įvesties. Vartotojas tai pat gali pasirinkti vieną iš 3 dangaus fonų (skybox), įjungti ir iš jungti muziką.
5. Svetimo kūno poveikis Saulės sistemai. Kaip ir HTML5, OpenGL ir Unreal Engine 4 projektuose, ši dalis leidžia įvesti svetimo kūno parametrus ir stebėti jo poveikį sistemai. Ši dalis tai pat yra skirta lyginimui tarp grafinio atvaizdavimo variklių.

3.14 Patarimai, pastebėjimai, rekomendacijos

Norint iškviesti funkcija iš struktūros, būtinai, išskyrus HTML5 projekte, reikia pereinamojo vektoriaus.

Aprašant naujas funkcijas struktūroje, jos nebūtinai turi būti gražinamos kaip vektoriaus koordinatės, priklausomai koks yra pradinis kreipinys.

Pradedančiajam, kuris norėtų išmokti dirbti su kokiu nors grafiniu varikliu rekomenduočiau pradžiai Unreal Engine 4, nes jis labai draugiškas pradedantiesiems su grafiniu modeliavimu ir programavimu. „Blueprint“ programavimo sistema leidžia laivai eksperimentuoti tiesiogiai su programos struktūra beveik be jokio kodo rašymo.

Rekomenduojma ir Unity grafinį žaidimų variklį, nors jis pradedančiajam yra sudėtingesnis nei UE4, nes daug ką reikia apipavidalinti skriptu. Jis tai pat yra draugiškas pradedančiajam su grafiniais varikliais. Dirbdamas su šiuo varikliu, asmeniškai jaučiau didesnę objektų valdymo kontrolę nei su UE4.

OpenGL Saulės sistemoje vietoj ***Tao.OpenGL.dll*** galima naudoti naujesnę ***OpenTK.dll***.

Unity grafinio variklio redaktoriuje kartais atsiranda normalizacijos (*normalize*) klaidos. Viena iš pagrindinių jos priežasčių yra susijusi su redaktoriaus rėžime esančia kamera, kai ji yra per toli nuo atvaizduojamo objekto. Ši klaida netrukdo projekto veikimui.

Išvados

1. Atlikus esamų virtualių Saulės sistemos modelių analizę galima teigti, kad didžioji dalis sukurtų virtualių Saulės sistemos modelių naudoja apskaičiuotas ir nekeičiamas dangaus kūnų orbitas, todėl šie modeliai netiko, išskyrus vartotojo sąsajos pavydžius, mano kuriamai virtualiai Saulės sistemai.

2. Atlikus matematinio modelio testavimą galima teigti kad jis yra teisingas, nes planetų padėtis apytiksliai atitinka pagal NASA astronomines lenteles.

3. Panaudojus matematini modelį *HTML5, OpenGL, Unity, Unreal Engine 4* grafiniuose varikliuose yra nustatyta, kad pasirinkti varikliai jam yra tinkami ir spėja pateikti skaičiuojamąsias reikšmes atvaizdavimui.

4. Atlikus projektų analizę geriausiai tolesniam projektų tobulinimui tinka *Unity* grafinis žaidimų variklis.

ANOTACIJA

Autorius: Andrius Galminas

Tema: *Virtualios Saulės sistemos kūrimas ir analizė.*

Šiaulių universitetas 2016.

Šiame darbe yra aprašoma virtualios Saulės sistemos kūrimas ir analizė naudojant Niutono visuotinės traukos dėsnis, mechanika ir skaitinius metodus. Sudarytas matematinis modelis yra pritaikytas *HTML5*, *OpenGL, Unity* ir *Unreal Engine 4* grafiniams varikliams. *Unity* grafinis žaidimų variklis buvo pasirinktas kaip pagrindinis variklis, kur buvo sudaryta „Žvaigždės kūrimo sistema“, kur vartotojas gali sukurti savo žvaigždės sistemą arba identišką Saulės sistemai ir išsaugoti ją.

SUMMARY

Author: Andrius Galminas

Subject: *Development and analysis of the Virtual Solar System.*

Siauliai University 2016.

This paper describes the Virtual Solar System Design and analysis using Newton’s universal law of gravity, mechanics and numerical methods. Mathematical model is adapted *HTML5, OpenGL, Unity* and *Unreal Engine 4* graphics engines. *Unity* graphical game engine has been chosen as the main engine, which using it „Žvaigždės kūrimo sistema“(Star development system) was created, where user can create own star system or identical to Solar System and save it.

Literatūra ir šaltiniai

1. NAVITSKAS Juozas. *FIZIKA I dalis, Mokomoji knyga*. Kaunas: „ARDIVA" leidykla, 2008. 17 – 19 p., 48 – 49 p. ISBN 978-9955-896-48-7. Prieiga per internetą: <http://asu.lt/wpcontent/uploads/2015/01/fizika_i_0.pdf>
2. Ažusienis A., Pučinskis A, Straižys V.. *Atronomija.* Vilnius: „Kultūros" leidykla, 2003. 69 – 74 p. ISBN 9986-435-10-2.
3. MUSIELAK Z.E., QUARLES B., 2015. *The three-body problem*. The University of Texas at Arlington, Arlington, TX 76019, USA. Prieiga per internetą: <http://arxiv.org/pdf/1508.02312.pdf> [Tikrinta 2016-05-14],
4. *JavaScript 3D library*. Prieiga per internetą: <https://github.com/mrdoob/three.js/> [Tikrinta 2016-05-14].
5. *OpenGL (Open Graphics Library).* Prieiga per internetą: <http://whatis.techtarget.com/definition/OpenGL-Open-Graphics-Library> [Tikrinta 2016-05-14].
6. *Glut - The OpenGL Utility Toolkit*. Prieiga per internetą: <https://www.opengl.org/resources/libraries/glut/> [Tikrinta 2016-05-14].
7. *The Tao Framework.* Prieiga per internetą: <https://sourceforge.net/projects/taoframework/>. [Tikrinta 2016-05-14].
8. *Solar Sistem Scope*. Prieiga per internetą: <http://www.solarsystemscope.com/> [Tikrinta 2016-05-14].
9. *3D Solar System Simulator*. Prieiga per internetą: <http://www.brightonastronomy.com/solarsystem.html> [Tikrinta 2016-05-14].
10. *Solar System Simulator*. Prieiga per internetą: <http://www.mpe.mpg.de/~sotiris/SolsarSystemSimulator.html> [Tikrinta 2016-05-14].
11. MAGHSOUDI Esfandiar. *3D Solar System Web*. Prieiga per internetą: <http://project-metis.com/SolarSystem/> [Tikrinta 2016-05-14].
12. SEVERINOV Nikita. *3D Solar System on WebGL*. Prieiga per internetą: <http://codecanyon.net/item/3d-solar-system-on-webgl/6552144> [Tikrinta 2016-05-14] ir <http://severinov.info/ru/> [Tikrinta 2016-05-14].
13. TSEREKH Vasily. *Solar system in 3D with opengl and C#.* Prieiga per internetą: <http://vasilydev.blogspot.lt/2011/08/solar-systems-in-3d-with-opengl-and-c.html> [Tikrinta 2016-05-14].

14. *The shortcomings of HTML5*. Prieiga per internetą: <http://rebuildingtheweb.com/en/html5-shortcomings> [Tikrinta 2016-05-14].

Informaciniai, papildomi šaltiniai

1. *Unreal Engine 4*. Prieiga per internetą: <https://www.unrealengine.com/what-is-unreal-engine-4> [Tikrinta 2016-05-14].

2. Hardware & Software Specifications(UE4). Prieiga per internetą: <https://docs.unrealengine.com/latest/INT/GettingStarted/RecommendedSpecifications/> [Tikrinta 2016-05-14].

3. *Unity*. Prieiga per internetą: <https://unity3d.com/> [Tikrinta 2016-05-14].

4. *System Requirements for Unity 5.3*. Prieiga per internetą: <https://unity3d.com/unity/system-requirements> [Tikrinta 2016-05-14].

5. *Recommended Hardware(UE4).* Prieiga per internetą: <https://wiki.unrealengine.com/Recommended_Hardware> [Tikrinta 2016-05-14].

6. AITKEN D. 2016. HTML-5-tutorial. Prieiga per internetą: <http://www.html-5-tutorial.com/about-html.htm> [Tikrinta 2016-05-14].

7. *16 Recommended 3D Game Engines*. Prieiga per internetą: <http://www.worldofleveldesign.com/categories/level_design_tutorials/recommended-game-engines.php> [Tikrinta 2016-05-14].

8. *Fizikos teorijos konspektas, Kinematikos pagrindai*. Prieiga per internetą: <https://protas.pypt.lt/fizika/fizikos_teorijos_konspektas/kinematikos_pagrindai> [Tikrinta 2016-05-14].

9. MARŠALKAITĖ G., VALIŪNAS M., 2014. *Astronomijos pratybų užduočių komplektas*. Vilnius. Prieiga per internetą: <http://www.lmnsc.lt/supadmin/kiti/lmitkcedit/uploads/files/AstronomijosPratybuUzduociuKomplektas.pdf> [Tikrinta 2016-05-14].

10. *Planetary Fact Sheet – Metric*. Prieiga per internetą: <http://nssdc.gsfc.nasa.gov/planetary/factsheet/> [Tikrinta 2016-05-14].

12. *Getting to Grips with HTML5 Browser Compatibility*. Prieiga per internetą: <https://speckyboy.com/2012/03/25/getting-to-grips-with-html5-browser-compatibility/> [Tikrinta 2016-05-14].

13. *Apsis*. Prieiga per internetą: <http://www.newworldencyclopedia.org/entry/Apsis> [Tikrinta 2016-05-14].

14. Jim Morrison. *HTML5 & CSS3 Support*. Prieiga per internetą: <http://blog.deepbluesky.com/blog/-/browser-support-for-css3-and-html5_72/> [Tikrinta 2016-05-14].

15. KOTOVYCH Oksana, BOWMAN C. John, 2015. *An Exactly Conservative Integrator for the n – Body Problem*. Department of Mathematical and Statistical Sciences, University of Alberta, Edmonton, Alberta, Canada T6G 2G1. Prieiga per internetą: <https://www.math.ualberta.ca/~bowman/publications/nbody.pdf> [Tikrinta 2016-05-14].
16. DUBSON M., 2013. *My Solar System*. Prieiga per internetą: <https://phet.colorado.edu/sims/my-solar-system/my-solar-system_en.html> [Tikrinta 2016-05-14].
17. *Gabūs ir Drasūs.* Prieiga per internetą: < http://gid.lt/> [Tikrinta 2016-05-14].

Priedai

1. Naudojami duomenys

Saulė: Masė – $1.989 \cdot 10^{30}\ kg$;

Lentelė 3 Naudojami planetų duomenys

Planetos pavadinimas	Atstumas nuo Saulės	Orbitos Greitis	Masė
Merkurijus	$6.9817079 \cdot 10^{10}\ m$	$38860\ m/s$	$3.302 \cdot 10^{23}\ kg$
Venera	$1.08941849 \cdot 10^{11}\ m$	$34784\ m/s$	$4.8685 \cdot 10^{24}\ kg$
Žemė	$1.52097701 \cdot 10^{11}\ m$	$29291\ m/s$	$5.97219 \cdot 10^{24}\ kg$
Marsas	$2.4922873 \cdot 10^{11}\ m$	$21972\ m/s$	$6.4185 \cdot 10^{23}\ kg$
Jupiteris	$8.16081455 \cdot 10^{11}\ m$	$12446\ m/s$	$1.899 \cdot 10^{27}\ kg$
Saturnas	$1.503983449 \cdot 10^{12}\ m$	$9137\ m/s$	$5.6846 \cdot 10^{26}\ kg$
Uranas	$3.006389405 \cdot 10^{12}\ m$	$6486\ m/s$	$8.6832 \cdot 10^{26}\ kg$
Neptūnas	$4.536874325 \cdot 10^{12}\ m$	$5385\ m/s$	$1.0243 \cdot 10^{26}\ kg$
Plutonas	$7.375927931 \cdot 10^{12}\ m$	$3676\ m/s$	$1.305 \cdot 10^{22}\ kg$

Lentelė 4 Naudojami palydovų duomenys

Palydovo pavadinimas	Atstumas nuo planetos	Orbitos Greitis	Masė
(Žemė) Menulis	$3.63104 \cdot 10^{8}\ m$	$968\ m/s$	$7.347673 \cdot 10^{22}\ kg$
(Marsas) Fobas	9376000 m	$2138\ m/s$	$1.0659 \cdot 10^{16}\ kg$
(Marsas) Deimas	$23470900\ m$	$1351.3\ m/s$	$1.8 \cdot 10^{15}\ kg$
(Jupiteris) Ijo	$423400000\ m$	$17263\ m/s$	$8.931938 \cdot 10^{22}\ kg$
(Jupiteris) Europa	$670900000\ m$	$13740\ m/s$	$4.799844 \cdot 10^{22}\ kg$
(Jupiteris) Ganimedas	$1071600000\ m$	$10868\ m/s$	$1.4819 \cdot 10^{23}\ kg$
(Jupiteris) Kalista	$1897000000\ m$	$8143 m/s$	$1.075938 \cdot 10^{23}\ kg$
(Uranas) Miranda	$129390000\ m$	$6660\ m/s$	$6.59 \cdot 10^{19}\ kg$
(Uranas) Arielis	$191020000\ m$	$5510\ m/s$	$1.353 \cdot 10^{21}\ kg$
(Uranas) Umbrelis	$266000000\ m$	$4670\ m/s$	$1.172 \cdot 10^{21}\ kg$
(Uranas) Titanija	$435910000\ m$	$3640\ m/s$	$3.527 \cdot 10^{21}\ kg$
(Uranas) Oberonas	$583520000\ m$	$3150\ m/s$	$3.014 \cdot 10^{21}\ kg$
`(Neptūnas) Tritonas	$354759000\ m$	$-4390\ m/s$	$2.14 \cdot 10^{22}\ kg$

Naudojami duomenys yra pagal Keplerio dėsnio tolimiausią, vidutinį atstumą ir mažiausią, vidutinį greitį modelio pradinėm objektų pozicijos koordinatėm ir poslinkiui sudaryti.

2. Skirtumas tarp mokamos ir nemokamos *Unity* versijos

Lentelė 5 Skirtumas tarp mokamos ir nemokamos Unity versijos

Kas įtraukta	Personal Edition	Professional Edition
Variklis su visomis savybėmis	Taip	Taip
Nėra autorinio mokesčio produktui (Royalty-free)	Taip	Taip
Visos platformos (yra ribota)	Taip	Taip
Pritaikoma ekrano užsklanda (Customizable Splash Screen)	Ne	Taip
Unity Cloud Build Pro - 12 menesių	Ne	Taip
Unity Analytics Pro	Ne	Taip
Komandos licenzija	Ne	Taip
Prioritetinis redaktoriaus klaidų(bug) taisymas	Ne	Taip
Žaidimo spartos ataskaitos	Ne	Taip
Beta versijų priėjimas	Ne	Taip
Neribotos pajamų įplaukos ir finansavimas produktui (Unlimited Revenue and Funding)	Ne	Taip
Įskaitytos platformos kurios bus dar tik naudojamos	Ne	Taip
Profesionalaus redaktoriaus ženklelio naudojimas	Ne	Taip
Asset Store Level 11	Ne	Taip
Source kodo prieiga (Source code access)	Ne	Papildomas mokestis
Premium Palaikymas (Premium Support)	Papildomas mokestis	Papildomas mokestis

3. Matematinis modelis

3.1 Planetos pradinės pozicijos skaičiavimas

Naudojama sferinė koordinačių sistema kuri yra aprašoma dviem kampais ir atstumu nuo koordinačių pradžios.

$$x = rcos\beta cos\alpha; \quad (7)$$

$$z = rcos\beta sin\alpha; \quad (8)$$

$$y = rsin\beta; \quad (9)$$

Čia

r – atstumas nuo koordinačių pradžios.

β – kampas tarp z ir y plokštumos.

α - kampas tarp x ir z plokštumos.

3.2 Pradinio greičio koordinatės sudarymas

Projekte yra du greičio koordinatės sudarymo metodai:

1. Poslinkio susiejimas su objekto pradine pozicija:

$$x = -vsin\alpha; \quad (10)$$

$$z = vcos\alpha; \quad (11)$$

$$y = 0;$$

Čia

v – objekto greitis;

α - kampas tarp x ir z plokštumos;

2. Tiesioginė įvestis į greičio x, y, z koordinates.

Kad būtų sudarytas trijų kūnų uždavinys, prie palydovo pradinių pozicijos ir greičio koordinačių pridedamos planetos pozicijos ir greičio koordinatės. Tokiu būdu yra sudaroma palydovo pozicija Saulės atžvilgiu.

3.3 Masės Centro formulės

Pagal apibrėžimą (žr. 7 p.) sudaromas masių centro koordinatės:

$$x_{MC} = \frac{1}{M} \sum_{i=1}^{n} x_i m_i \tag{12}$$

$$y_{MC} = \frac{1}{M} \sum_{i=1}^{n} y_i m_i \tag{13}$$

$$z_{MC} = \frac{1}{M} \sum_{i=1}^{n} z_i m_i \tag{14}$$

Čia

M – Visų objektų masių suma;

x, y, z – pradinės objekto koordinatės;

m – objekto masė;

i – sumuojamo objekto koordinatė ir jos sandauga su mase;

n – objektų kiekis;

x_{MC}, y_{MC}, z_{MC} – Masių centro koordinatės;

Sudarytas masių centro koordinates atimame iš visų esamų objektų pradinių koordinačių, sudarydami naujas pradines koordinates objektams.

$$x = x - x_{MC}; \tag{15}$$

$$y = y - y_{MC}; \tag{16}$$

$$z = z - z_{MC}; \tag{17}$$

3.4 Pagreitis

Pagreičiui skaičiuoti, kai masių centras nėra naudojamas, yra naudojama Niutono dviejų kūnų uždavinio (žr. 1 pav.) pagreičio formulės:

$$a_x = \frac{-G(M+m)(x-x2)}{r^3} \quad (18)$$

$$a_y = \frac{-G(M+m)(y-y2)}{r^3} \quad (19)$$

$$a_z = \frac{-G(M+m)(z-z2)}{r^3} \quad (20)$$

Kur pagal Euklido normą randamas ilgis r:

$$r = \sqrt{(x-x2)^2 + (y-y2)^2 + (z-z2)^2} \quad (21)$$

x, y, z – kintamo objekto koordinatės;

$x2, y2, z2$ – objektas kuris veikia kintamą objektą;

M – kintamo objekto x, y, z masė;

m – objekto $x2, y2, z2$ masė;

G – gravitacijos konstanta;

Kai naudojamas masių centras, pagreičio formulėje kintamo objekto masė M nebenaudojama, nes ji jau buvo prisumuota (12), (13) ir (14) formulėse:

$$a_x = \frac{-G \cdot m(x1-x2)}{r^3} \quad (22)$$

$$a_y = \frac{-G \cdot m(y1-y2)}{r^3} \quad (23)$$

$$a_z = \frac{-G \cdot m(z1-z2)}{r^3} \quad (24)$$

3.5 Nauja objekto pozicija esant tolygiai kintamam judėjimui

Nauja pozicija apskaičiuojama naudojant tolygiai kintamo judėjimo formulės, naudojamos mechanikoje:

$$x_{n+1} = x_n + v_{x_n} t + \frac{a_x t^2}{2}; \quad (25)$$

$$y_{n+1} = y_n + v_{y_n} t + \frac{a_y t^2}{2}; \quad (26)$$

$$z_{n+1} = z_n + v_{z_n} t + \frac{a_z t^2}{2}; \quad (27)$$

Čia

x, y, z – objekto koordinatės;

v_x, v_y, v_z – objekto poslinkio koordinatės;

a_x, a_y, a_z – objekto pagreičio koordinatės;

t – laiko tarpas;

n – skaičiavimo žingsnis.

3.6 Objekto (tolygiai kintamo) greičio projekcija

Naujas greitis apskaičiuojamas naudojant tolygiai kintamo judėjimo formules, kur prie esamos greičio koordinatės pridedamas pagreičio koordinatės:

$$v_{x_{n+1}} = v_{x_n} + a_x t; \quad (28)$$

$$v_{y_{n+1}} = v_{y_n} + a_y t; \quad (29)$$

$$v_{z_{n+1}} = v_{z_n} + a_z t; \quad (30)$$

Čia

v_x, v_y, v_z – greičio koordinatės;

a_x, a_y, a_z – pagreičio koordinatės;

4. DVD disko turinys

DVD diske yra įrašytos sukurtos Saulės Sistemos modeliai su *HTML5, OpenGL, Unity, Unreal Engine 4* grafiniais varikliais. *HTML5* ir *OpenGL* aplankale tai pat yra įrašyti autorių projektų originalai kurie buvo naudojami esamų projektų sudarymui.

5. Vartotojo instrukcija

5. 1 *Unity* Saulės sistemos valdymas

Tab - laisva kamera;

Q - 2d kamera/2d camera;

P - Sustabdyti ir paleisti simuliaciją;

Esc - Meniu;

Laisvos (3d) kameros valdymas:

W, kairės pelės mygtukas - judėjimas į priekį;

S, dešinės pelės mygtukas - judėjimas atgal;

A - judėjimas kaire puse;

D - judėjimas dešine puse.

Kamera yra jautri pelės pozicijai.

2D Kameros valdymas:

W - judėjimas aukštyn;

S - judėjimas žemyn;

A - judėjimas kaire puse;

D - judėjimas dešine puse.

Kairės pelės mygtukas - judėjimas į priekį;

Dešinės pelės mygtukas - judėjimas atgal;

Metų įvestyje vartotojas gali įvesti metus, mėnesį, dieną ir planetų pozicijos apytiksliai bus sudėliotos palei tuos metus. Neįvedus nieko, planetos yra sudėliotos palei 2016/01/15.

Laiko(s) lauką yra įvedamas laiko tarpas, kuris yra naudojas kiekvienu žingsniu esamų objektų pozicijai, pagreičiui ir greičiui perskaičiuoti t.y. kiek objektai pajudėjo praėjus įvestam laikui. Rekomendacija įrašyti nuo **1 iki 100**, nes didesnis kuo didesnis skaičiui įrašomas, tuo labiau atvaizduojamų objektų orbitų tikslumas išsibalansuoja.

Atvaizdavimo Ciklo lauke yra įvedamas ciklas kuris paspartina atvaizduojamųjų objektų atvaizdavimo greitį. Rekomendacija įvesti yra iki 2000. Priklausomai nuo naudojamo PC galingumo, galima įvesti ir didesnį kiekį.

Objektų atvaizdavimo greitis yra lygus ***Atvaizdavimo Ciklas*** padaugintas iš ***Laikas(s)***:

Atvaizdavimo greitis = Atvaizdavimo Ciklas * Laikas(s).

Vartotojas tai pat gali pasirinkti vieną iš 3 dangaus paveikslėlių (skybox) ir paleisti arba išjungti muziką.

Saulės sistemos informacinėje dalyje palipdomai yra naudojami viršutiniai klaviatūros mygtukai, kur judėjimas yra stebimas parsiniktos planetos atžvilgiu.

1 - Merkurijus;

2 - Venera;

3 - Žemė;

4 - Maras;

5 - Jupiteris;

6 - Saturnas;

7 - Uranas;

8 - Neptūnas;

9 – Plutonas.

Čia kameros yra jautrio pelės pozicijai ir keliauja kartu su parinkta planeta. Vartotojas šių kamerų judinti negali. Norint grįžti prie laisvos 3d kameros – Tab, 2d kameros – Q.

Kai simuliacija su P mygtuku sustabdyta, atsiranda pelė ir paspaudus esamus objektus atsiranda meniu langas parodantis tos planetos informaciją.

Šioje programos dalyje vartotojas gali sukurti savo arba identišką Saulės sistemai virtualų modelį, atitinkamai įvesdamas Žvaigždės masę, planetų masę, greitį, atstumą ir palydo masę, greitį, atstumą.

GreitisX, GreitisY, GreitisZ yra objekto judėjimo pasirinkimo kryptys. **Y** yra aukščio koordinatė. Norit naudotis jais į **Orbitos greitis objekto pozicijoje** lauką turi būti įrašytas 0 arba nieko, nes kitu atveju jie bus neužskaityti.

Rašant duomenis vietoj kablelio turi būti dedamas taškas, kitaip įvedami duomenys bus nenuskaitomi.

Pavyzdys 1:

Žvaigždės meniu:

Žvaigždės masė – $2 * 10^{30}$ kg;

Spindulio dydis – 300. (Čia pasirenkamas objekto atvaizduojamas dydis);

Atvaizdavimo greitis – 1 iki 6000. (Priklausimai nuo PC galimybių, gali būti ir daugiau).

Laikas – 30 s. (Nuo 1 iki 100);

Planetos meniu:

Planetų kiekis – 1. (Galima iki 20 planetų);

Palydovų kiekis – 1. (Galima iki 4 palydovų kiekvienai planetai, jei planeta be palydovų – įvesti 0);

Pasirinkite planetą – 1. (Čia yra pasirenkama planeta, jei norima įvesti arba keisti jos parametrus)

Planetos masė – $6 * 10^{24}$ kg.

Atstumas – $1.5 * 10^{11}$ m. (Dėl gravitacijos konstantos, atstumas yra skaičiuojamas metrais).

Orbitos greitis objekto pozicijoje – 30000 m/s;

Horizontali Pozicija – 0 (horizontalioje plokštumoje objektą galima pastatyti 360^0 kampu);

Vertikali Pozicija – 0 (vertikalioje plokštumoje objektą galima pastatyti 360^0 kampu);

Planetos dydis – 200.

Palydovo meniu:

Pasirinkite palydovą – 1. (palydovo pasirinkimas yra atskiras kiekvienai planetai);

Palydovo masė – $7.3 * 10^{22}$ kg.

Atstumas – $4 * 10^8$ m.

Orbitos greitis objekto pozicijoje – 968 m/s;

Horizontali Pozicija – 0;

Horizontali Pozicija – 0;

Palydovo dydis – 100.

Vartotojas tai pat gali pasirinkti tekstūrą Žvaigždei, kiekvienai planetai, palydovui, viena iš 4 dangaus paveikslėlių ir įjungti išjungti muziką.

Saulės sistema su svetimo kūno įvestimi

Čia vartotojas esamai sistemai įveda svetimą kūno parametrus, panašiai kaip ir *Žvaigždės Sistemos Kūrime.* Saulės sistemos planetos susidėlioja pagal įvestus metus, mėnesį ir dieną. Svetimą kūną reprezentuoja kubas.

Pavyzdys 2:

Laikas – 60 s;

Ciklas – 1 iki 6000; (Jei vaizdas strikinėja, reikia įvesti mažesnį skaičių, nei yra pateikta)

Masė – $6 * 10^{28}$ kg;

Atstumas – $1.8 * 10^{11}$ m.

HorKampas – 0;

VerKampas – 0;

GreitisX – 30000;

GreitisY – 0;

GreitisZ – 0;

5. 2 HTML5 Saulės sistemos valdymas

Valymas:

P - Objekto Meniu, tęsti simuliaciją;

Tab - Keitimas tarp įvesties laukų;

Kameros Valdymas:

W, Kairės pelės mygtukas - judėjimas į priekį;

S. Dešinės pelės mygtukas - judėjimas atgal;

A - Judėjimas į kairę;

D - Judėjimas Į dešinę;

Q - Kameros pakreipimas kaire puse;

E - Kameros pakreipimas dešine puse;

R - Kameros judėjimas aukštyn;

F - Kameros judėjimas žemyn;

Kamera yra jautri pelės pozicijai.

Planetos yra sustatytos pagal 2016/01/01 pozicijas.

Svetimo kūno įvesčiai patikrinti galima naudoti **Pavyzdys 2** duomenis. Dėl HTML5 savybių **Ciklo** įvestis gali būti įvedamas žymiai didesnis nei pateikiama pavyzdyje, viskas prikiauso nuo PC galimybių. Įvedus Ciklą ir Laiką ir objekto parametrų įvesties laukus, kad jie veiktų, reikia paspausti patvirtinimo mygtukus.

5. 3 OpenGL Saulės sistemos valdymas

Esc – Išeiti programą;
P – Objekto meniu/tęsti simuliaciją;
Kairės pelės mygtukas - judėjimas į priekį;
Dešinės pelės mygtukas - judėjimas atgal.

Kamera yra jautri pelės pozicijai.
Planetos yra sustatytos pagal 2016/01/01 pozicijas.

Svetimo kūno įvesčiai patikrinti galima naudoti **Pavyzdys 2** duomenis. Kadangi sistema yra labiau apkraunama **Ciklo** įvesti patariama daryti nuo 1 iki 1000, o **Laiką** 1 iki 100 s. Įvedus Ciklą ir Laiką ir objekto parametrų įvesties laukus, kad jie veiktų, reikia paspausti patvirtinimo mygtukus.

5. 4 Unreal Engine 4 Saulės sistemos valdymas

Esc – Išeiti programą;
P – Objekto meniu/tęsti simuliaciją
W - judėjimas į priekį;
S - judėjimas atgal;
A - Judėjimas į kairę;
D - Judėjimas Į dešinę;

Kamera yra jautri pelės pozicijai.
Planetos yra sustatytos pagal 2016/01/01 pozicijas.

Svetimo kūno įvesčiai patikrinti galima naudoti **Pavyzdys 2** duomenis. **Ciklo** įvestyje patariama naudoti nuo 1 iki 6000, priklausomai nuo PC galimybių. Įvedus Ciklą ir Laiką ir objekto parametrų įvesties laukus, kad jie veiktų, reikia paspausti patvirtinimo mygtukus.