

PÉCSI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR
MATEMATIKAI ÉS INFORMATIKAI INTÉZET

NAPRENDSZER SZIMULÁCIÓS PROGRAM UNITY-BEN



Témavezető: KISS-VINCZE TAMÁS
tanársegéd

Szerző: FARKAS PATRIK MÁRK (BYONIZ)
PROGRAMTERVEZŐ
INFORMATIKUS BSC

Pécs, 2024

„Képzeljük el, hogy a "világ" állandóan mozgásban lévő tárgyak bonyolult elrendeződése, egyetlen hatalmas sakkjátszma, az istenek játsszák, s mi csak megfigyelhetjük a játék szabályait. E világméretű játszma szabályai: a fizika alapjai. Viszont ha az összes szabályt ismernénk is, akkor sem lennének képesek megérteni, miért pont a megfigyelt sakkhúzásra került sor a játszmában ez már túlságosan bonyolult, és értelmünk véges. A sakkozók jól tudják, hogy a szabályok megtanulása nem nehéz feladat, mégis gyakran nehéz a legjobb lépést kiválasztani, vagy megérteni, hogy egy játékos az adott helyzetben miért éppen azt húzza, amit húz. Mindez a természet nagy sakkjátszmájára sokkalta inkább érvényes.”

(Richard P. Feynman)

Kivonat

Jelen munka a Naprendszer kiválasztott égitesteinek keringési szimulációjáról szól. Bemutatásra fog kerülni egy általam készített, a Unity játékmotorban írt szimulációs program és annak működése. Eme kvázi játék háttérkódjainak a megértéséhez nemcsak informatikai, hanem fizikai és matematikai ismeretekre is szükség van, amiket a dolgozatban igyekszem rendelkezésre bocsátani. Ennek függvényében a hangnem és a struktúra váltakozó, valahol ismeretterjesztő, néhol inkább szakmai hangvételű. A projekt célja, az önnön kíváncsiságom ki-elégítésén kívül, egy látványos tanító program elkészítése a fiatalabb generációk számára. Habár a "látványos" egy szubjektív kifejezés, ezalatt egy interaktív és háromdimenziós programot értek, ami emellett informatív is. Habár a realisztikus ábrázolásra törekedtem, hamar rá kellett jönnöm, hogy ezen a téren csak részleges sikereket tudok elérni. A szimulációs modellem alapját a Newton-féle gravitációs törvény, a tömegpontok mozgása és a negyedrendű Runge-Kutta módszer adja. Az utóbbira nagyobb hangsúlyt fektettem, hisz ez áll közel a szakomhoz, illetve ez teljesen új információ volt számomra is. A következő fejezetekben ezeket fogom boncolgatni.

Kulcsszavak: Naprendszer, szimuláció, C#, Unity, RK4, 2-test

Tartalomjegyzék

I. Bevezetés	5
II. Fizikai háttér	6
II.1. Pár alapfogalom	6
II.2. A csillagászat történelme	6
II.3. Kepler-törvények	7
II.3.1. Kepler-törvények	7
II.4. Newton-féle gravitációs törvény	8
II.4.1. Newton-féle gravitációs törvény	8
II.4.2. 2-test probléma	8
II.4.3. n-test probléma	9
II.4.4. A tömegpontok mozgása	9
III Matematikai háttér	10
III.1 A Runge-Kutta módszer	10
III.1.1 Az RK4 módszer általános alakja	10
IV.A Solar System Orbit Simulation program	16
IV.1.A Unity játékmotor	16
IV.2.A szimulációról	17
IV.2.1. Egyszerűsítések	17
IV.2.2.A szimulációs képletek	17
IV.3.A felhasznált adathalmaz	18
IV.4.A szoftver működéséről	19
IV.4.1. Telepítési útmutató	19
IV.4.2.A méretarányosság	20
IV.4.3.Az égitest objektumok	21
IV.4.4.A UI és UX	22
IV.4.5.A lépésköz és a perspektíva váltó csúszkák	24
IV.4.6.A zoomolás	25
IV.4.7.A fókuszálás mechanika	25
IV.4.8.Az információs kártyák	26
IV.5.Javítási és továbbfejlesztési javaslatok	27
IV.5.1.Javítandó hibák	27
IV.5.2.Továbbfejlesztési javaslatok, ötletek	27
V. Összegzés	29
Szakirodalom jegyzék	30

I. Bevezetés

Szakdolgozatom témájának az égitestek keringésének szimulációját választottam. Az égitestek elliptikus, hiperbolikus vagy parabolikus pályán keringenek egy vagy több központi égitest befolyása szerint. Ezek szimulációja rendkívül fontos szerepet játszik az úrkutatásban.

Témám választásának 3 oka van, és ezeket szeretném most jobban kifejteni. Az első ok a csillagászat iránti szeretetem, ami egészen gyerekkoromig visszavezethető. Még anyai nagypapám nevelte belém azzal, hogy ilyen témajú könyveket vett nekem ajándékba, illetve egy amatőr csillagász távcsövet, mellyel sokszor hajnalig az eget fürkészük együtt. A második indok egy kicsit az előzőhez köthető, mert szeretném, ha a nálam fiatalabb generációkat jobban érdekelné a csillagászat. Úgy érzem, hogy habár egy bizonyos fokig őket is foglalkoztatja a téma, viszont általában elriadnak a fizika miatt. Szeretném, ha ezen a szimulációs program meghozná a kedvüket ahhoz, hogy többet olvassanak ezekről.

A harmadik és egyben utolsó ok a kíváncsi természetemből és az ezzel járó makacságomból ered. Másodéves koromban beadandó projektként szerettem volna írni egy Python programot, amely a Naprendszerünk bolygóinak mozgását realisztikusan szimulálta volna. Sajnos elég gyorsan elakadtam, ezért az interneten kezdtem el hasonló szoftvereket keresni. A találatok közül a legtöbb azt az egyszerűsítést alkalmazta, hogy a bolygók pályáját körformájúnak tekintette. Ezt én azért furcsálltam, hisz általános iskolától kezdve azt tanítják a diákoknak, hogy a bolygók pályája ellipszis formájú. Illetve a műholdak pályáját is valahogy ki kell számolni, amiknek a pályája merőben eltér a körtől. Ezekből kiindulva biztos voltam abban, hogy kell lennie egy vagy akár több módszernek is arra, hogy ezeket a mozgásokat szimulálni tudjuk. Emiatt elkezdtem kutatni, és arra kellett rájönnöm, hogy ez sokkal mélyebb téma, mint azt először gondoltam és a megoldás minden, csak nem triviális. Ezenkívül fontos megemlítenem, hogy a legtöbb program, amit találtam kettő dimenzióban vizsgálta a problémát. Tehát, a következő célokot tűztem ki magamnak:

1. A programnak látványosnak kell lennie, hogy a fiatalokat érdekelje, illetve 3 dimenziósnak, hogy különböző szemszögekből lehessen megfigyelni a égitestek keringését.
2. A szimuláció tartalmazza a Napot, a Naprendszer bolygóit és azok holdjait.
3. Az égitestek keringése legyen realisztikus.
4. A méretek legyenek arányosak.

A szakdolgozatomban szó lesz a téma megértéséhez szükséges fizikai és matematikai háttéről, illetve arról, hogy ezeket hogyan implementáltam egy, a *Unity* játékmotor segítségével írt szimulációs programban.

II. Fizikai háttér

Ehhez a fejezethez, a [1],[2],[3],[4],[5],[6],[7] és a [8] forrásokat használtam.

II.1. Pár alapfogalom

Mielőtt még mélyebbre ássuk magunkat ebben a témaban, érdemes pár alapfogalmat letisztázni:

- égitest: Égitestnek nevezünk minden olyan természetes vagy mesterséges objektumot, mely a világűrben található.
- csillag: Olyan saját fénnnyel rendelkező égitestek, melyek plazmából épülnek fel; formájuk szferoid. Csak saját gravitációs terük tartja őket egyben. Saját fényét nukleáris folyamatok melléktermékeként termeli. A Földhöz legközelebbi csillag a Nap.
- bolygó: Olyan jelentősebb tömegű égitest, mely egy csillag körül kering. A Nap bolygórendszerében 8 található, távolság szerint növekvő sorrendben: Merkúr, Vénusz, Föld, Mars, Jupiter, Szaturnusz, Uránusz és Neptunusz. Ezeket két csoportra lehet bontani kőzetbolygó vagy gázórias.
- hold: Olyan jelentősebb tömegű égitest, mely egy bolygó körül kering. A Föld csak egy holddal rendelkezik - a Holddal -, de más bolygók rendelkezhetnek többel is vagy eggyel sem.
- Naprendszer: Naprendszer alatt a Napot és a körülötte keringő kisebb-nagyobb testek összességét tekintjük. A Naprendszer a Nap gravitációs vonzása tartja egyben.[5]

II.2. A csillagászat történelme

Amióta az emberiség több ezer évvel ezelőtt először felnézett az égboltra, az óta szeretné megfejteni összes titkát. Eleinte isteneknek gondolták őket, akik a magasból néznek le ránk, figyelve minden mozdulatunkat. Később kialakult a csillagászat, az emberiség egyik legősibb tudománya.

Az égitestek mozgása rendkívüli fontossággal bírt elődjeink számára. Ebből tudták megállapítani például, hogy mikor kell elvetni a magokat, hogy időben le tudják azokat aratni. Eleinte geocentrikusnak hitték a világot, azaz a Föld körül kering minden, amit az égbolton meg lehet figyelni. Kopernikusz (Toruń, Lengyelország 1473. – Frombork, Lengyelország 1543.) lengyel csillagász állt elő

a heliocentrikus világképpel, mely szerint a Nap (görög: Helios) a Világ minden-ség középpontja. Habár ma már tudjuk, hogy csak a mi Naprendszerünk középpontja a Nap, de akkoriban ez óriási felfedezésnek számított, mely erőteljesen szembement az egyház által támogatott geocentrikus világképpel. [3] Az égitestek keringésével rengeteg tudós foglalkozott, közülük a teljesség igénye nélkül kiemelném Galieo Galileit, Tycho Brahét, Johannes Keplert és Isaac Newton-t.

II.3. Kepler-törvények

Johannes Kepler (Weil der Stadt, Német-római Birodalom, 1571. – Regensburg, Német-római Birodalom, 1630.) német származású matematikus és csillagász rendkívül fontos szerepet játszott az égi mechanika kutatásában. 1596-os *Mysterium Cosmographicum* című könyvében elsőként próbálta megmagyarázni fizikai úton a bolygók pályáját egy, a Napból kiáradó erővel.[6]

II.3.1. Kepler-törvények

1600-ban együtt dolgozott egy másik remek csillagással, Tycho Brahéval. Képességeik egymást kiegészítették, Brahe jó megfigyelő volt, míg Kepler páratlan matematikai képességekkel rendelkezett. Brahe halála után Keplerre hagyta megfigyeléseit. Ezek alapján fogalmazta meg törvényeit:

1. minden bolygó elliptikus pályán mozog a Nap körül, amelynek egyik fókuszpontja a Nap, képlete

$$r = \frac{l}{1 + e \cdot \cos \varphi} \quad (\text{II.3.1.1})$$

ahol (r, φ) a bolygók napközpontú polárkoordinátái, l a fókuszon átmenő, a nagytengelyre merőleges húr fele, e pedig az excentricitás.

2. A Naptól bolygókhoz húzott sugárvektor azonos idő alatt azonos területet súrol.
3. Bármely két bolygó keringési idejének négyzete az ellipszispályák fél nagytengelyeinek köbeivel arányos; ismertebb formájában:

$$\frac{T_1^2}{T_2^2} = \frac{a_1^3}{a_2^3} \quad (\text{II.3.1.2})$$

Ahol, T_1 és T_2 a bolygók keringési idejei, az a_1 és a_2 a fél nagytengelyei.[1]

Habár ezek mindegyike nagyon fontos csillagászati törvény, mégis ezek közül számunkra a szimulációhoz legfontosabb az első. Mint azt korábban, a bevezetésben is említettem, az egész projektemet az indította el, hogy nem akartam a bolygók pályáját körnek tekinteni. Habár ez a pálya bizonyos megfelelő körül-mények között akár kör formájú is lehet, viszont ez rendkívül ritka. Nemcsak a bolygók pályájáról mondható ez el, hanem a holdakéről is. Egyéb égitesteknek (pl. üstökös, aszteroida stb.) a pályája hiperbolikus is lehet, viszont ezekre ez a szakdolgozat, illetve a szimuláció sem fog kitérni.

II.4. Newton-féle gravitációs törvény

Sir Isaac Newton (Woolsthorpe Manor, Anglia, 1642.- Kensington, Egyesült Királyság, 1727.) angol polihisztor, a történelem egyik legkiemelkedőbb tudósa volt. A Newton-féle gravitációs törvény azt mondja ki, hogy bármely két test kölcsönösen vonzza egymást.

II.4.1. Newton-féle gravitációs törvény

Két pontszerűnek tekinthető test között ez az erő egyenesen arányos a tömegek szorzatával, valamint fordítottan arányos a köztük lévő távolság négyzetével. Eme törvényét a Philosophiae Naturalis Principia Mathematica művében publikálta 1687. július 5-én.[8][1]

$$F_g = G \frac{m_1 m_2}{r^2} \quad (\text{II.4.1.1})$$

ahol:

- F_g [N] a gravitációs erő,
- G a gravitációs állandó, aminek az értéke : $6.6743 \cdot 10^{-11} m^3 kg^{-1} s^{-2}$,
- m_1 és m_2 [kg] a két test tömege,
- r [m] két közötti távolság

II.4.2. 2-test probléma

A 2-test probléma elemzi két pontszerű tömeg gravitációs kölcsönhatását, amit Newton-féle gravitációs (lásd: II.4.1), illetve Newton második törvénye (a dinamika alaptörvénye: $a = F/m$) ír le. A megoldások között szerepelnek az ellipszisek, hiperbolák és parabolák, melyek fókuszpontja a tömegközéppontban van:

$$a(r) = -\frac{F(r)}{m} = -\frac{GM}{||\vec{r}||^2} \hat{r} \quad (\text{II.4.2.1})$$

ahol $a(t)$ a pillanatnyi gyorsulás m/s^2 -ben, G a gravitációs állandó, $||\vec{r}||$ a pozíciójuk vektori különbségének - azaz az égitest lokális pozíció vektorának - hossza, \hat{r} az r vektor normalizált alakja, a m a kisebbik és a M a nagyobbik tömegpont.

A 2-test probléma a klasszikus mechanika egyik közismert, megoldással rendelkező problémája. A 2-test probléma pontos és előrejelezhető eredményeket ad két test egymásra gyakorolt kölcsönhatásáról, amelyek ideális esetben izoláltak más külső behatásuktól. Eme modellt használják a műholdak pályájának tervezésében, bolygók mozgásának szimulációjában és az űrhajózásban.

II.4.3. n-test probléma

Az n -test probléma, a 2-test problémához hasonlóan testek egymásra gyakorolt hatására szeretne pontos eredményt adni n darab test esetén, ahol $n > 2$. Ez szintén a klasszikus mechanika egy híres megoldatlan problémája. Nem létezik általános zárt formájú megoldás, mert a testek pályáját sokkal nehezebb előre jelezni a kölcsönhatások összetettsége és a rendszer kaotikus természete miatt.

II.4.4. A tömegpontok mozgása

A szimulációhoz használt képletekhez szükségünk lesz a már korábban megbeszélt Newton-féle gravitációs képlet mellett még a tömegpontok mozgásának egyenleteire:[4]

$$v(t) = \int_0^t a(\tau) d\tau \quad (\text{II.4.4.1})$$

$$r(t) = \int_0^t v(\tau) d\tau \quad (\text{II.4.4.2})$$

Ahol:

- $a(t)$ [m/s^2] a pillanatnyi gyorsulás
- $F(t)$ [N] a tömegpontra ható erők pillanatnyi eredője
- m [kg] a tömegpont tömege
- $v(t)$ [m/s^{-1}] a tömegpont pillanatnyi sebessége
- $r(t)$ [m] a tömegpont pillanatnyi pozíciója

III. Matematikai háttér

III.1. A Runge-Kutta módszer

Ennek a fejezetnek az elkészítéséhez a [9], [10], [11] és a [12] forrásokat használtam.

Carl Runge és Wilhelm Kutta olyan matematikusok voltak, akik jelentős hatással bírtak a numerikus analízis területére, különösen azon módszerek fejlesztésében, amelyek a differenciálegyenletek numerikus megoldásait szolgálják.

Carl David Tolmé Runge (Bréma, Német Szövetség, 1856 – Göttingen, Német Birodalom, 1927) német matematikus és fizikus volt, aki jelentős munkát végzett az analízis és a numerikus matematika területén.[10]

Martin Wilhelm Kutta (Byczyna, Felső Szilézia, 1867 – Fürstenfeldbruck, Bajorország, 1944) szintén német matematikus volt, aki leginkább azért vált ismertté, mert kidolgozta a Runge-Kutta módszereket (1900 körül).[11]

A Runge-Kutta módszer egy numerikus módszer, amelyet differenciálegyenletek, különösen kezdeti érték problémák megoldására használnak. A módszer különböző változatai lehetővé teszik, hogy nagyobb pontossággal és stabilitás-sal közelítsük a differenciálegyenletek megoldását, mint az egyszerűbb Euler-módszer.

A legismertebb változata a negyedrendű Runge-Kutta módszer, gyakran csak "RK4" néven emlegetik. Az RK4 módszer négy különböző "slope" vagy meredekség kiszámítását használja az egyenlet közelítő megoldásának kiszámításához egy adott lépésközben. Ezeket a következő lépés számításához használják fel a differenciálegyenlet megoldásának közelítésére. Ez a módszer népszerű, mert jó kompromisszumot nyújt a számítási erőforrásigény és a pontosság között. Széles körben használják modellek és szimulációk készítéséhez.

A szimuláció tervezésekor felmerült bennem alternatívának az ötödrendű Runge-Kutta (RK5), illetve a Runge-Kutta-Fehlberg (RKF45) módszer is. Ezeket viszont hamar elvetettem attól tartván, hogy az RK4 számolásigénye is meg fogja haladni a számomra rendelkezésre álló erőforrásokat, nemhogy a nála még erő-forrásigényesebb RK5 és RKF45.

III.1.1. Az RK4 módszer általános alakja

Ez az alfejezet a [12] szakirodalom alapján lett kidolgozva. minden Runge-Kutta típusú módszer alapja az, hogy az y megoldásfüggvény x_{n+1} és x_n pontbeli érté-kének különbségét a következő alakban fejezzük ki:

$$y_{n+1} - y_n = \sum_{i=1}^m w_i k_i, \quad (\text{III.1.1.1})$$

ahol a w_i mennyiségek állandóak és

$$k_i = h_n f \left(x_n + \alpha_i h_n, y_n + \sum_{j=1}^{i-1} \beta_{ij} k_j \right) \quad h_n = x_{n+1}, \alpha_1 = 0. \quad (\text{III.1.1.2})$$

A Runge-Kutta típusú módszereknél a lépésközöt lépésenként változtathatjuk, ezért van szükség a h_n jelölésre. Nyilván ha a w_i , α_i és β_{ij} állandók ismertek, akkor a (III.1.1.1) formula önállóan elindítható módszer az elsőrendű differenciálegyenlet kezdetiérték-feladatának numerikus megoldására.

Célunk a w_i , α_i és β_{ij} állandókat úgy megválasztani, hogy a (III.1.1.1) képlet bal és jobb oldalának az (x_n, y_n) pont körüli Taylor-sorában h_n^r megfelelő együtthatói $r = 1$ valamely $r = M$ -ig rendre megegyezzenek. M nem lehet m -nél nagyobb. Az ebből kapott képletet M -ed rendű Runge-Kutta formulának hívjuk. Kényelmi szempontból nem fogunk megkülönböztetést tenni az igazi és a számított megoldás jelölése között (tehát pl.: az y függvényt differenciálni fogjuk, mintha helyes megoldás lenne).

A képlet bal oldalának kifejtése:

$$y_{n+1} - y_n = \sum_{t=1}^{\infty} h_n^t y_n^{(t)} \frac{1}{t!}. \quad (\text{III.1.1.3})$$

A $dy/dx = f(x, y)$, $y(x_0) = y_0$ differenciálegyenletből

$$\begin{aligned} y_n^{(t)} &= \frac{d^{t-1}}{dx^{t-1}} f(x_n, y_n) = \left(\frac{\partial}{\partial x} + \frac{dy}{dx} \frac{\partial}{\partial y} \right)^{t-1} f(x_n, y_n) = \\ &= \left(\frac{\partial}{\partial x} + f \frac{\partial}{\partial y} \right)^{t-1} f(x_n, y_n), \end{aligned} \quad (\text{III.1.1.4})$$

ahol $f = f(x, y)$. Ennek a felhasználásával a (III.1.1.3) képletből

$$y_{n+1} - y_n = \sum_{t=0}^{\infty} \frac{h_n^{t+1}}{(t+1)!} \left(\frac{\partial}{\partial x} + f \frac{\partial}{\partial y} \right)^{t-1} f(x_n, y_n) \quad (\text{III.1.1.5})$$

lesz.

Hozzunk létre egy D operátort, ami

$$D = \frac{\partial}{\partial x} + f_n \left(\frac{\partial}{\partial y} \right), \quad f_n = (x_n, y_n) \quad (\text{III.1.1.6})$$

és így például írhatjuk azt, hogy

$$\left[\frac{\partial}{\partial x} + f \frac{\partial}{\partial y} \right]^2 f(x_n, y_n) = D^2 f + f_n Df|_n, \quad (\text{III.1.1.7})$$

ahol a $|_n$ azt jelenti, hogy az összes mennyiséget az (x_n, y_n) pontban kell venni (és

h -t h_n -nel helyettesítjük).

A (III.1.1.7) azonossággal a (III.1.1.5) kifejtés első néhány tagja a következő:

$$\begin{aligned} y_{n+1} - y_n = & \left[h f + \frac{h^2}{2!} Df + \frac{h^3}{3!} (D^2 f + f_y Df) + \right. \\ & + \frac{h^4}{4!} (D^3 f + f_y D^2 f + f_y^2 Df + 3Df Df_y) + \frac{h^5}{5!} (D^4 f + \\ & + 6Df D^2 f_y + 4D^2 f Df_y + D^2 f f_y^2 + Df f_y^3 + 3(Df)^2 f_{yy} + D^3 f f_y + \\ & \left. + 7f_y Df Df_y \right] |_n + O(h_n^6). \end{aligned} \quad (\text{III.1.1.8})$$

A (III.1.1.1) formula jobb oldalának kifejtéséhez először a kétváltozós függvények Taylor-sorának kifejezését használjuk fel:

$$f \left[x_n + \alpha_i h_n, y_n + \left(\sum_{j=1}^{i-1} \beta_{ij} \right) h_n f_n \right] = \sum_{j=1}^{\infty} h_n^t D_i^t f(x_n, y_n) \frac{1}{t!}, \quad (\text{III.1.1.9})$$

ahol

$$D_i = \alpha_i \frac{\partial}{\partial x} + \left(\sum_{j=1}^{i-1} \beta_{ij} \right) f_n \frac{\partial}{\partial y}. \quad (\text{III.1.1.10})$$

A (III.1.1.9) és a (III.1.1.10) egyenlet felhasználásával a (III.1.1.1) jobb oldalán álló k_i mennyiségeket ki tudjuk fejteni. Mivel $\alpha_1 = 0$, így

$$k_1 = h_n f_n. \quad (\text{III.1.1.11})$$

A k_2 mennyiségre a következőt kapjuk:

$$\begin{aligned} k_2 = h_n f(x_n + \alpha h_n, y_n + \beta_{21} k_1) &= h_n f(x_n + \alpha_2 h_n, y_n + \beta_{21} h_n f_n) = \\ &= h_n \sum_{t=0}^{\infty} h_n^t D_2^t f(x_n, y_n) \frac{1}{t!}, \end{aligned} \quad (\text{III.1.1.12})$$

ahol a D_2 operátor jelentése a (III.1.1.10) képlet szerinti. A k_3 kifejtésére (III.1.1.11) és (III.1.1.12) felhasználásával a következőképpen járunk el:

$$\begin{aligned} k_3 &= h_n f(x_n + \alpha_3 h_n, y_n + \beta_{31} k_1 + \beta_{32} k_2) = \\ &= h_n f[x_n + \alpha_3 h_n, y_n + (\beta_{31} + \beta_{32}) h_n f_n + \beta_{32} (k_2 - h_n f_n)] = \\ &= h_n f \sum_{t=0}^{\infty} \left[h_n D_3 + \beta_{32} (k_2 - h_n f_n) \frac{\partial}{\partial y} \right]^t f(x_n, y_n) \frac{1}{t!} \end{aligned} \quad (\text{III.1.1.13})$$

A (III.1.1.13) képletet tekinthetjük a k_3 -nak h_n hatványai szerint haladó kifejtésének, ha a (III.1.1.13) kifejezésben a k_2 -t a (III.1.1.12) formulával helyettesítjük. Az előbbi eljárás pedig bármely k_i kifejtésére általános módszerként alkalmazható. Legyen

$$\begin{aligned}
k_i &= h_n f \left(x_n + \alpha_i h_n, y_n + \sum_{j=1}^{i-1} \beta_{ij} k_j \right) = \\
&= h_n f \left\{ x_n + \alpha_i h_n, y_n + \sum_{j=1}^{i-1} [\beta_{ij} h_n f_n + \beta_{ij} (k_j - h_n f_n)] \right\} = \quad (\text{III.1.1.14}) \\
&= h_n \sum_{t=0}^{\infty} \left[h_n D_i + \sum_{j=2}^{i-1} \beta_{ij} (k_j - h_n f_n) \frac{\partial}{\partial y} \right]^t f(x_n, y_n) \frac{1}{t!}.
\end{aligned}$$

Ezután helyettesítsük be a k_j , ahol ($j < i$), mennyiségek helyébe korábban a kapott, h_n hatványok szerint haladó kifejtésüket. A tagokat h_n^5 -ig megtartva az eredmény $i \in \{1, 2, 3, 4\}$ -re:

$$k_1 = h f|_n, \quad (\text{III.1.1.15})$$

$$k_2 = h f + h^2 D_2 f + \frac{h^3}{2!} D_2^2 f + \frac{h^4}{3!} D_2^3 f + \frac{h^5}{4!} D_2^4 f|_n + O(h_n^6), \quad (\text{III.1.1.16})$$

$$\begin{aligned}
k_3 &= h f + h^2 D_3 f + h^3 \left(\frac{1}{2!} D_3^2 f + \beta_{32} h_y D_2 f \right) + h^4 \left(\frac{1}{6} D_3^3 f + \right. \\
&\quad \left. + \frac{\beta_{32}}{2} f_y D_2^2 f + \beta_{32} D_2 f D_3 f_y \right) + h^5 \left[\frac{1}{24} D_3^4 f + \frac{\beta_{32}}{6} f_y D_2^3 f + \right. \\
&\quad \left. + \frac{\beta_{32}}{2} D_2^2 f D_3 f_y + \frac{\beta_{32}^2}{2} f_{yy} (D_2 f)^2 + \frac{\beta_{32}}{2} D_2 f D_3^2 f_y \right] |_n + O(h_n^6), \quad (\text{III.1.1.17})
\end{aligned}$$

$$\begin{aligned}
k_4 &= h f + h^2 D_4 f + h^3 \left(\frac{1}{2} D_4^2 f + \beta_{42} f_y D_2 f + \beta_{43} f_y D_3 f \right) + \\
&\quad + h^4 \left[\frac{1}{6} D_4^3 f + \frac{1}{2} \beta_{42} f_y D_2^2 f + \beta_{32} \beta_{43} (f_y)_2^D f + \frac{1}{2} \beta_{43} f_y D_3^2 f + \right. \\
&\quad \left. + \beta_{42} D_2 f D_4 f_y + \beta_{43} D_3 f D_4 f_y \right] + h^5 \left[\frac{1}{24} D_4^4 f + \frac{1}{6} \beta_{42} f_y D_2^3 f + \right. \\
&\quad \left. + \frac{1}{2} \beta_{32} \beta_{43} (f_y)^2 D_2^2 f + \beta_{32} \beta_{43} f_y D_2 f D_3 f_y + \frac{1}{6} \beta_{43} f_y D_3^3 f + \right. \\
&\quad \left. + \frac{1}{2} \beta_{42} D_4 f_y D_2^2 f + \frac{1}{2} \beta_{43} D_4 f_y D_3^2 f + \frac{1}{2} \beta_{42}^2 + f_{yy} D_2^2 f + \right. \\
&\quad \left. + \beta_{42} \beta_{43} f_{yy} D_2 f D_3 f + \frac{1}{2} \beta_{43}^2 f_{yy} D_3^2 f + \frac{1}{2} \beta_{42}^2 f_y + \right. \\
&\quad \left. + \frac{1}{2} \beta_{43} D_3 f_4^2 f_y + \beta_{43} \beta_{32} f_y D_2 f D_4 f_y \right] |_n + O(h_n^6). \quad (\text{III.1.1.18})
\end{aligned}$$

Negyedrendig bezárólag minden Runge-Kutta típusú formula levezetését lehetővé teszi (ez csaknem minden gyakorlati problémához elegendő pontossággal bír) a (III.1.1.15)-(III.1.1.18) képletecsoport.

Helyettesítsük a (III.1.1.15)-(III.1.1.18) formulákat a (III.1.1.1) képlet jobb ol-

dalába, a (III.1.1.8) kifejtést bal oldalába; rendre egyeztessük a h_n azonos hatványainak együtthatóját minden oldalon; így azt kapjuk, hogy

$$h_n : \quad w_1 + w_2 + w_3 + w_4 = 1, \quad (\text{III.1.1.19})$$

$$h_n^2 : \quad w_2 D_2 f + w_3 D_3 f + w_4 D_4 f = \frac{1}{2!} D f, \quad (\text{III.1.1.20})$$

$$\begin{aligned} h_n^3 : \quad & \frac{1}{2} (w_2 D_2^2 f + w_3 D_3^2 f + w_4 D_4^2 f) + f_y [w_3 \beta_{32} D_2 f + \\ & + w_4 (\beta_{42} D_2^2 f + \beta_{43} D_3^2 f)] = \frac{1}{3!} (D^2 f + f_y D f), \end{aligned} \quad (\text{III.1.1.21})$$

$$\begin{aligned} h_n^4 : \quad & \frac{1}{6} (w_2 D_2^3 + w_3 D_3^3 + w_4 D_4^3) + \frac{1}{2} f_y [w_3 \beta_{32} D_2^2 f + \\ & + w_4 (\beta_{42} D_2^2 f + \beta_{43} D_3^2 f)] + [w_3 \beta_{32} D_2 f D_3 f_y + w_4 (\beta_{42} D_2 f D_4 f_y + \\ & + \beta_{43} D_3 f D_4 f_y)] + [w_4 \beta_{32} \beta_{43} (f_y)^2 D_2 f] = \frac{1}{4!} (D^3 f + f_y D^2 f + \\ & + 3 D f D f_y + f_y^2 D f). \end{aligned} \quad (\text{III.1.1.22})$$

Ha azt szeretnénk, hogy a w_i, α_i , illetve β_{ij} együtthatók $f(x, y)$ -tól függetlenek legyenek - ami szükséges ahhoz, hogy a formula használható legyen a gyakorlatban -, akkor a fenti egyenletek négy helyett valójában nyolc feltételt jelentenek. Ugyanis, a (III.1.1.21) és (III.1.1.22) egyenlet bal oldalán a szögeletes zárójelben álló mennyiségeknek, amelyek az operátorokban homogén kifejezések, meg kell egyezniük a jobb oldali megfelelő mennyiségekkel. Ezenfelül, ha az eredményül adódó nyolc egyenlet $f(x, y)$ -tól függetlenül érvényes akkor a

$$\frac{D_j f}{D f} \quad (j = 2, 3, 4) \quad \text{és} \quad \frac{D_j f_y}{D f_y} \quad (j = 3, 4) \quad (\text{III.1.1.23})$$

arányoknak állandóknak kell lenniük. Ez utóbbi feltételrendszer teljesül, ha

$$\alpha_i = \sum_{j=1}^{i-1} \beta_{ij} \quad (i = 2, 3, 4), \quad (\text{III.1.1.24})$$

mert akkor

$$D_i = \alpha_i D. \quad (\text{III.1.1.25})$$

Végül is tehát a nyolc feltételi egyenlet a következő lesz:

$$\begin{aligned} & w_1 + w_2 + w_3 + w_4 = 1, \\ & w_2 \alpha_2 + w_3 \alpha_3 + w_4 \alpha_4 = \frac{1}{2}, \\ & w_2 \alpha_2^2 + w_3 \alpha_3^2 + w_4 \alpha_4^2 = \frac{1}{3}, \\ & w_3 \alpha_2 \beta_{32} + w_4 (\alpha_2 \beta_{42} + \alpha_3 \beta_{43}) = \frac{1}{6}, \end{aligned}$$

$$w_2\alpha_2^3 + w_3\alpha_3^3 + w_4\alpha_4^3 = \frac{1}{4}, \quad (\text{III.1.1.26})$$

$$w_3\alpha_2\beta_{32} + w_4(\alpha_2\beta_{42} + \alpha_3\beta_{43}) = \frac{1}{6},$$

$$w_2\alpha_2^3 + w_3\alpha_3^3 + w_4\alpha_4^3 = \frac{1}{4},$$

$$w_3\alpha_2^2\beta_{32} + w_4(\alpha_2^2\beta_{42} + \alpha_3^2\beta_{43}) = \frac{1}{12},$$

$$w_3\alpha_2\alpha_3\beta_{32} + w_4\alpha_4(\alpha_2\beta_{42} + \alpha_3\beta_{43}) = \frac{1}{8},$$

$$w_4\alpha_2\beta_{32}\beta_{43} = \frac{1}{24}, \quad (\text{III.1.1.27})$$

ahol az első egyenlet a (III.1.1.19)-nek, a második a (III.1.1.20)-nak, a következő kettő a (III.1.1.21)-nek, és végül az utolsó négy a (III.1.1.22)-nek felel meg. A (III.1.1.24) és a (III.1.1.26) eredő feltételrendszer 11 egyenletet jelent 13 ismeretlenre, és így általában az ismeretlenekre megoldathtó, sőt két szabadsági fok is marad. Mejegyezzük, hogy az utolsó egyenlet miatt a h_n^4 rendű pontosság elérésére a k_4 tagot szükségképpen meg kell tartanunk. Ezzel igazoltuk a korábbi állítást miszerint $M \leq m$.

A leggyakrabban használt RK4 módszer az, amelyben $\alpha_2 = \alpha_3 = 1/2$; ennek a módszernek a Butcher táblája a

0				
1/2				
1/2				
1				

IV. A Solar System Orbit Simulation program

Eme fejezetben a [13],[14],[15],[16],[17],[18],[19][20],[21] és [22] forrásokat használtam.

Ez a szekció a szimulációs programról fog szólni, amit (nem túl kreatívan) *Solar System Orbit Simulation*-nek (alternatív megnevezés: SSOS) neveztem el. A programot a *Unity* játékmotor segítségével készítettem el, ami rendkívül erős renderelő képességekkel rendelkezik. A *Unity* elsődlegesen támogatott nyelve a *C#*, így én is ezt használtam a legtöbb szkript megírásához. A kezdőadatok manipulációjához pedig *Python*-t használtam. A *C#* és *Unity* számomra teljesen ismeretlen vizek voltak, a projekt elkezdésekor.

IV.1. A Unity játékmotor

A *Unity* egy játékmotor, amit a *Unity Technologies* nevű cég fejleszt. Első verzióját 2005-ben adták ki, azóta rengeteg változásban és fejlesztésen esett át. A *Unity* egy rendkívül erős, valós idejű fejlesztő platform, ami azt jelenti, hogy a szerkesztő programban eszközölt változások rögtön megjelennek a fejlesztő előtt.

Nagy népszerűségnek örvend renderelő képességei és fejlesztőbarát felülete miatt, ezért sok különböző területen használják; a 2D-s videójátékoktól kezdve, a VR (virtális valóság) fejlesztésen át, még az egyik Mercedes Benz autó virtuális asziszten्�csének a 3D-s grafikájához is. Népszerűségének egy el nem hanyagolható oka az, hogy az ehhez hasonló játékmotorokkal ellentétben egy sokak által kedvelt programozási nyelvet támogat, a *C#*-ot (pl.: a *Godot* a saját *GDScrip* nyelvét támogatja), de emelett a vizuális programozásra is lehetőség van.[13][22]

Egy *Unity* projekt általában több, de minimum egy *Scene*-ből épül fel, ezekre én néha menüként fogok hivatkozni. Későbbiekben majd részletesebben kifejtem, de az én programom 4 *Scene*-ből épül fel (*Main Menu*, *User Guide*, *Cards* és *Solar System Simulation*). minden *Scene* rendelkezik egy fő kamerával, aminek a neve *Main Camera*.[21][13]

A legtöbb általam írt osztály a *MonoBehaviour* alaposztályt használja, ami azt jelenti, hogy nem létezhet önmagában, mindenkorban egy objektumhoz van kötve komponensként. Ez az alaposztály nagyon sok fontos függvényt létesít számunkra, de érdemes kiemelni a "Start" és az "Update" metódusokat. Az első az objektum létrejöttekor lefut egyszer, míg a másik képkockánként (*frame*) fut le, ciklikusan.

A projektem elkezdése előtt nem foglalkoztam még a *Unity*-vel, illetve a *C#*-pal is csak érintőleges találkoztam, emiatt nagy kihívás volt számomra ez a program.

IV.2. A szimulációról

IV.2.1. Egyszerűsítések

Az égitestek mozgásának vizsgálatához és a szimuláció számolásigényének lecsökkentéséhez az alábbi egyszerűsítéseket tesszük:

- Az égitesteket pontszerűnek tekintjük.
- A szimulációban eltekintünk a perturbációtól, azaz izoláltnak tekintjük az égitesteket, ezzel lebontván az n-test problémát, sok 2-test probléma szimulációjára.

IV.2.2. A szimulációs képletek

A negyedrendű Runge-Kutta módszer (lásd: III.1.1), a Newton-féle gravitációs törvény (lásd: II.4.1) és a tömegpontok mozgásképleteinek (lásd: II.4.4) segítségével megkaphatjuk a szimulációt, kvázi "szívét" jelentő formulákat:

$$\begin{aligned}
 \vec{a}(\vec{r}(t_n)) &= -\frac{GM}{||\vec{r}(t_n)||^2} \hat{r}(t_n) \\
 \vec{k}_{1v} &= \vec{a}(\vec{r}(t_n)) dt \\
 \vec{k}_{1r} &= \vec{v}(t_n) dt \\
 \vec{k}_{2v} &= \vec{a}\left(\vec{r}(t_n) + \frac{1}{2}\vec{k}_{1r}\right) dt \\
 \vec{k}_{2r} &= \left[\vec{v}(t_n) + \frac{1}{2}\vec{k}_{1v}\right] dt \\
 \vec{k}_{3v} &= \vec{a}\left(\vec{r}(t_n) + \frac{1}{2}\vec{k}_{2r}\right) dt \\
 \vec{k}_{3r} &= \left[\vec{v}(t_n) + \frac{1}{2}\vec{k}_{2v}\right] dt \\
 \vec{k}_{4v} &= \vec{a}\left(\vec{r}(t_n) + \vec{k}_{3r}\right) dt \\
 \vec{k}_{4r} &= \left[\vec{v}(t_n) + \vec{k}_{3v}\right] dt \\
 \vec{v}(t_{n+1}) &= \vec{v}(t_n) + \frac{1}{6}(\vec{k}_{1v} + 2\vec{k}_{2v} + 2\vec{k}_{3v} + \vec{k}_{4v}) \\
 \vec{r}(t_{n+1}) &= \vec{r}(t_n) + \frac{1}{6}(\vec{k}_{1r} + 2\vec{k}_{2r} + 2\vec{k}_{3r} + \vec{k}_{4r})
 \end{aligned} \tag{IV.2.2.1}$$

Ahol,

- $\vec{r}(t_n)$ a tömegpont relatív pillanatnyi pozíció vektora a nagyobbik égitesthez képest
- $||\vec{r}(t_n)||$ az $r(t_n)$ vektor hossza vagy normálformája $||\vec{r}(t_n)|| = \sqrt{x^2 + y^2 + z^2}$
- $\hat{r}(t_n)$ az $r(t_n)$ vektor normalizált alakja $\hat{r}(t_n) = \vec{r}(t_n) \cdot ||\vec{r}(t_n)||^{-1}$
- $M [kg]$ a nagyobbik égitest (tömegpont) tömege

- G a gravitációs állandó, aminek az értéke : $6.6743 \cdot 10^{-11} m^3 kg^{-1} s^{-2}$,
 - $m [kg]$ a vizsgált égitest (tömegpont) tömege
 - $\vec{a}(\vec{r}(t_n)) [m/s^2]$ a gyorsulás, a pillanatnyi pozíció vektor függvényében
 - $\vec{F}(\vec{r}) [N]$ a tömegpontra ható erők eredője, a pillanatnyi pozíció vektor függvényében
 - $\vec{v}(t_n) [m/s^{-1}]$ a tömegpont pillanatnyi sebességvektora
 - $\vec{k}_{iv} [m/s^2]$ az RK4 i -edik slope-ja a sebességfüggvény közelítéséhez, ($i \in 1, 2, 3, 4$)
 - $\vec{k}_{ir} [m]$ az RK4 i -edik slope-ja a pozíció közelítéséhez, ($i \in 1, 2, 3, 4$)
 - dt a lépésköz
 - $\mathbf{v}(t_{n+1}) [m/s^2]$ az égitest következő sebessége
 - $\mathbf{r}(t_{n+1})$ az égitest következő pozíciója

IV.3. A felhasznált adathalmaz

A szimulációhoz mindenképpen valós adatokat szerettem volna felhasználni, ezért egy internetes API-t (*Application Programming Interface*, azaz alkalmazás programozási interfész) kerestem ehhez. Ez nehezebbnek bizonyult, mint azt először gondoltam. Több hónapos keresés után a <https://api.le-systeme-solaire.net/en/> ingyenes API mellett döntöttem, viszont ez se volt tökéletes.

Az adatok manuális szűrése és tisztítása sok munkába tellett volna, ezért erre a célra *Python* nyelven írtam szkripteket. Megszabadultam a fölösleges és a hiányzó adatoktól, néhány helyen az utóbbit sikerült imputálni. Ennek következtében egy letisztult JSON fájlhoz jutottam. Az eredeti 366 felvett égitestből 177 került be a szimuláció által használt fájlba. Demonstráció gyanánt a Nap-Föld-Hold hármost használom az alábbi dokumentum részletben:

```
1 {"stars": [{  
2             "id": 1,  
3             "name": "Sun",  
4             "mass": 1.989e+30,  
5             "vol": 1.412e+18,  
6             "meanRadius": 695508.0}, ...]  
7     "planets": [...{  
8                 "id": 8,  
9                 "name": "Earth",  
10                "mass": 5.97237e+24,  
11                "vol": 1083210000000.0,  
12                "aroundObjectId": 1,  
13                "semiMajorAxis": 149598023,  
14                "perihelion": 147095000,
```

```

15         "aphelion": 152100000,
16         "eccentricity": 0.0167,
17         "inclination": 0.0,
18         "meanRadius": 6371.0084}, ...],
19     "moons": [...{
20         "id": 10,
21         "name": "Moon",
22         "mass": 7.346e+22,
23         "vol": 928430000000.0,
24         "aroundObjectId": 8,
25         "semiMajorAxis": 384400,
26         "perigee": 363300,
27         "apogee": 405500,
28         "eccentricity": 0.0549,
29         "inclination": 5.145,
30         "meanRadius": 1737.0, ...}]}

```

Mint ahogy az a fájlban is látható, a különböző típusú égitestek hasonló adatokkal rendelkeznek, néhány egyedi kivételével. Mindegyiknek van egy egyszerű, egész szám típusú id-je, ami egyedi. Az égitestek tömegét (mass) kilogramm-ban, térfogatukat (vol) köbkilométerben tároljuk. Ez utóbbi néhány holdnál hiányzik, viszont, mivel a számolásban nem játszik szerepet, így ez nem probléma. A távolsághoz köthető adatok (semiMajorAxis, perihelion, aphelion, perigee, apogee), illetve az égitest sugara (meanRadius) kilométerben van megadva. Az excentricitás (eccentricity) megmutatja, hogy egy égitest pályája mennyire tér el a körtől; a kör excentricitása 0. Az inklináció (inclination) egy égitest pályájának, a koordináta-rendszer alapsíkjával bezárt szöge. A bolygók és a holdak rendelkeznek egy aroundObjectId nevű mezővel, ami megmutatja, hogy melyik égitest körül kering, melyik rendszerhez tartozik. Ez nagyon fontos lesz a szimulációban.

IV.4. A szoftver működéséről

IV.4.1. Telepítési útmutató

Mielőtt még rátérnék a szoftver működésére, először a telepítéséhez szeretnék segítséget nyújtani. Az "SSOS Setup.exe" letöltése után le kell futtatni azt, ami automatikusan elindítja a telepítő varázslót. A számunkra megfelelő opcionális kiválasztása után az SSOS készen áll a használatra. Ha a felhasználó meg nem változtatja, akkor az alapértelmezett mappája a "C:\Users\{felhasználó neve}\AppData\Local\Programs\SSOS". A felhasználó eldöntheti, hogy szeretne-e asztali ikont¹ a programhoz vagy sem. Ha a felhasználó úgy dönt, hogy nincs szüksége már a programra, akkor a mappája tartalmaz egy "unins000.exe" nevű eltávolító alkalmazást, ami futtatáskor letörli a telepített fájlokat.

¹Az ikon ingyenesen letölthető innen: <https://www.rawpixel.com/image/7573765/png-sticker-planet>

Ezt a varázslót egy *Inno Setup Compiler* nevű programmal készítettem el, ami nagyban megkönnyítette a dolgomat. Annak előnyét, hogy a felhasználók között csak ezt a telepítő alkalmazást kell elterjeszteni, az egész projekt helyett, szerintem nem kell magyaráznom.[14]

IV.4.2. A méretarányosság

Ez számomra egy nagyon fontos pont volt, a kitűzött céljaim között. Sajnos a fejlesztés közben szembe kellett néznem a ténnel, hogy más fontosabb célja-im megvalósításával ütközik a méretarányosság megvalósítása, ezért el kellett engednem.

A méretarányos ábrázolással sok hónapot foglalkoztam, de végül csak részleges sikereket értem el. Egy nagyon fontos leckét sikerült ebből megtanulnom: az úr üres. Ez az olvasó számára rendkívül bugytának hangozhat. Ugyanis, amíg az ember nem látja a számokat, addig eklatánsnak gondolja ezt. Viszont, ha valaki kicsit is jobban beleássa magát ebbe a téma rájön, hogy sokkal üresebb, mint azt képzelte. Vegyük példának a Nap-Föld távolságot, ahhoz képest, hogy mekkora a két égitest:

- A Nap átmérője: $\sim 1\ 391\ 016\ km$
- A Föld átmérője: $\sim 12\ 742\ km$
- A köztük lévő távolság (perihéliumkor): $\sim 147\ 095\ 000\ km$

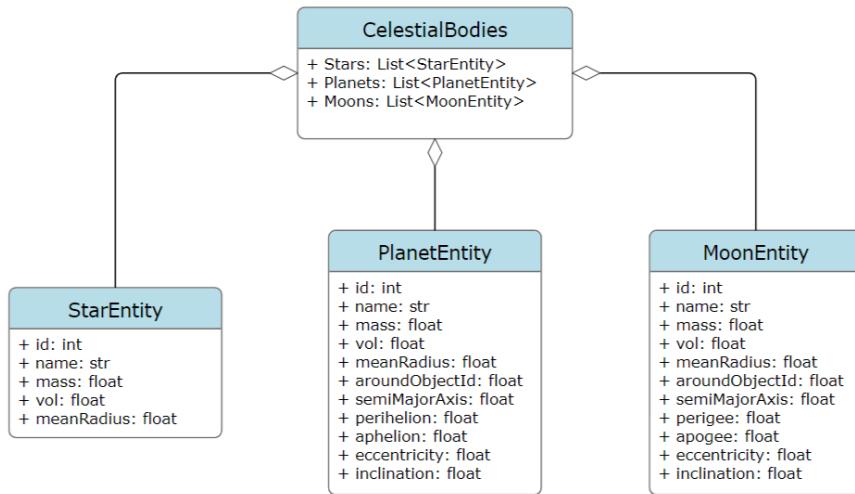
Eszerint a Nap átmérője nagyjából 109-szer nagyobb, mint a Földé, illetve a köztük lévő legkisebb távolság a Föld átmérőjének kb. 11 544.12, míg a Nap átmérőjének kb. 105.75-szorosa. Ha az égitestek méretét és a köztük lévő távolságot egységesen szeretnénk skálázni, akkor vagy semmit nem látnánk, vagy a túl nagy méretek miatt a játékmotor nem tudná renderelni azt. Arról nem is beszélve, hogy a Föld csak a harmadik bolygó a nyolcból, illetve a holdról még nem is volt szó.

Ezeket a problémákat a következő rendszerrel próbáltam orvosolni. Egyik skálázási alapomnak a csillagászati egységet, vagyis a Nap és a Föld közötti átlagos távolságot vettetem. Ezt jelezzük AU-val (az angol *astronomical unit*-ból) értéke pedig: $AU=149\ 597\ 871\ km$. A bolygók közti távolságokat az AU-tal osszuk el, majd azt egy tetszőleges számmal beszorozzuk (különben a távolságok túl kicsik lennének), ez legyen 12. A holdknál megkeresem a rendszer közepét jelentő bolygó legközelebbi holdját (már ha rendelkezik holddal) és annak a távolságával osztom el a többi holdét, majd beszorozom 8-cal.

Az égitestek mérete valójában nem fontos a szimulációban, hisz amúgy is pontszerűnek tekintjük őket. Ezért a Nap egy egység sugarú gömb, a bolygók méret szerint növekvő sorrendbe 0.1 egységtől indulnak és 0.1-esével nőnek. A holdakat, főleg az idő hiánya miatt, úgy írtam meg, hogy méretük fele akkora legyen, mint a bolygójuk.

IV.4.3. Az égitest objektumok

A CelestialBodiesLoader nevű osztályom egy planetLoader nevű üres objektumhoz van hozzákötve, ami lefutáskor a korábban bemutatott JSON fájlt először sztring formájúvá alakítja, majd egy Newtonsoft nevű könyvtár JsonConvert osztályának DeserializeObject függvényének segítségével egy CelestialBodies nevű sajátosztályom példányává teszi. Ez a CelestialBodies osztály a következő képpen néz ki:



1. ábra. A CelestialBodies, a StarEntity, a PlanetEntity és a MoonEntity osztályok osztálydiagramja

Mint látható az 1. ábrán, a CelestialBodies valójában a 3 égitest típus egy-egy osztályának példányaiiból készített listát tartalmaz.[19]

A CelestialBodiesLoader 2 másik függvényt foglal magába: a LoadSunAndPlanets és a LoadMoonsOfPlanet-et. Az előbbi a planetLoader létrejöttetekor szintén lefut, s a következőképpen működik.

Első körben, végig iterál a Stars listán. Ha rendelkezik saját textúrával, akkor azzal hoz létre egy objektumot, ha nem akkor az alap fehér gömbbel. Ezután az objektumhoz tartozó Star komponens nevét és tömegét állítja be.

Még ezen a cikluson belül végigfut a Planets, leellenőrzi, hogy az adott példány aroundObjectId-ja megegyezik-e az aktuális csillagéval. Ha igen, akkor megkeresi, hogy rendelkezik-e textúrával és ha igen, akkor azzal hozza létre az objektumot, ha nem, akkor az alap fehér textúrával. Ezután lekéri az objektum Planet komponensét és beállítja a különböző adatait.

A holdak nem jelennék meg a szimulációban, amikor elindul, csak ha a felhasználó rákattint a bolygójára és fókuszba kerül (lásd: IV.4.7). A LoadMoonsOfPlanet lefutásakor a MoonEntity listából csak a megfelelő bolygó holdjai fognak megjelenni, amiben megint csak a bolygó id-jának az aroundObjectId-val való összehasonlítása segít. Ezután lekéri az objektum Moon komponensét és beállítja a megfelelő értékeit. A Föld Holdján kívül minden hold textúrája fehér, mert a csomag nem tartalmazta a többet.

Az objektumokat az egyszerű kezelhetőség kedvéért létrehozáskor címkekkel

(tag) láttuk el. A bolygó objektumok "Planet", a hold objektumok "Moon" és a csillag objektumok, ami egyenlőre csak a Nap, "Star" tag-gel láttuk el. Ez később hasznunkra fog válni, amikor az összes azonos címkével ellátott objektummal szeretnénk dolgozni vagy például ha meg szeretnénk állapítani, hogy milyen objektumra kattintot a felhasználó.

A szimuláció kezdetekor (a Nap kivételével) minden égitest pozícióját a gravitációs középpontjuk legközelebbi, leskálázott értékére ($peri_{sim}$) állítjuk. Az egyszerűség kedvéért kezdéskor az y és a z koordinátákat 0-nak tekintjük, x pedig legyen egyenlő $peri_{sim}$ -mel, azaz

$$\vec{r} = \begin{bmatrix} peri_{sim} \\ 0 \\ 0 \end{bmatrix} \quad (\text{IV.4.3.1})$$

Ahol, \vec{r} a lokális pozíciója az égitestnek. Miután minden érték be lett állítva, elindul az Update függvény frame-nkénti ciklikus meghívása, ami pedig tartalmazza a szimulációs függvények hívását. Ezek a függvények a már korábban megtárgyalt szimulációs képletecsoport (lásd.: (IV.2.2.1) formulák) alapján lettek megírva.

A már korábban sokat említett textúrákat a *Unity Asset Store*-ból töltöttem le. Az asset neve: *Planets of the Solar System 3D*². Ez a csomag ingyenes volt, ehhez képest részletesen kidolgozott textúrákat tartamaz és nagyban hozzájárult ahhoz, hogy látványos legyen a projektem. Például: minden égitest rendelkezik 3 LOD-dal (Level Of Detail, azaz részletességi szint), ami a kamera közelsgéjtől függ, azaz minél közelebb van annál részletesebb a textúra.

IV.4.4. A UI és UX

A felhasználói felület elkészítéséhez két ingyenes asset-et használtam. A cso-magok nevei: *Dark Theme UI*³ és *Simple Button Set 02*⁴, amikben nagyon sok hasznos ikont és dizájnt találtam, amiket fel tudtam használni. A tervezésnél törekedtem a modern, letisztult stílus mellett a látványosságra.

²<https://assetstore.unity.com/packages/3d/environments/planets-of-the-solar-system-3d-90219>

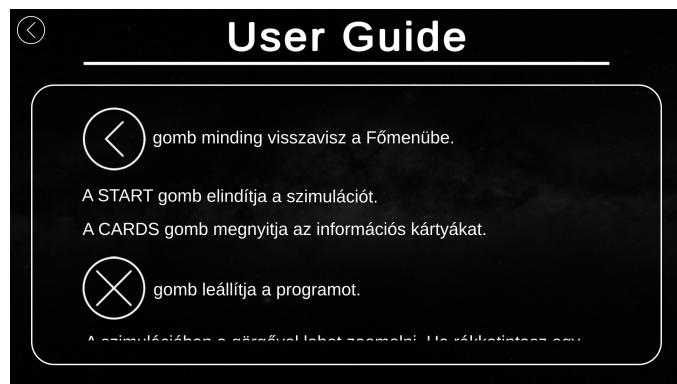
³<https://assetstore.unity.com/packages/2d/gui/dark-theme-ui-199010>

⁴<https://assetstore.unity.com/packages/2d/gui/icons/simple-button-set-02-184903>



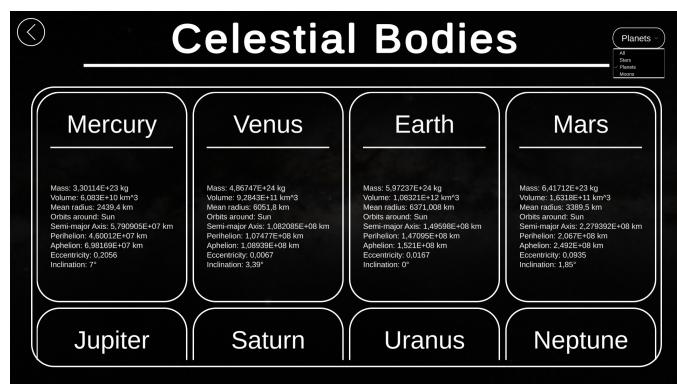
2. ábra. A Main Menu, innen lehet eljutni a program többi részéhez

A felhasználók – az alkalmazás megnyitása után – automatikusan főmenübe (Main Menu) kerülnek (lásd: 2. ábra). A felhasználók innen tovább léphetnek a felhasználói útmutatóhoz (az "i" gombbal), a "START" gombbal a szimulációba, a "CARDS" gombbal az információs kártyákhoz vagy az "X"-szel bezárhatják az alkalmazást.



3. ábra. A felhasználó útmutató, használat előtt ajánlott elolvasni

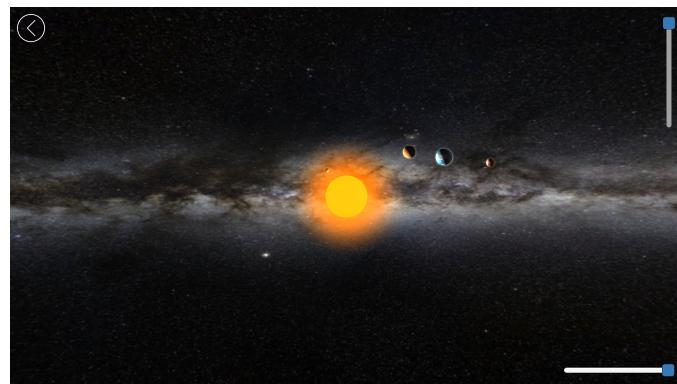
A felhasználói útmutatót (User Guide, lásd: 3. ábra) erősen ajánlott elolvasni az első használat előtt. A görgővel vagy az érintőpaddal lehet lefele görgetni benne. Ezt egy képszerkesztő alkalmazással készítettem el. Innen a bal felső sarokban található gombbal lehet visszalépni a Main Menu-be.



4. ábra. Az információs kártyák menü, a bolygók szűrő bekapcsolása után

A kártyákról majd egy későbbi fejezetben (lásd: IV.4.8) szeretnék részletesebben írni. A Cards menüben lehet az információs kártyák között szűrni, az égitest típusa szerint (lásd: 4. ábra): All (Összes), Stars (Csillagok), Planets (Planéták) és Moons (Holdak). Ezen az oldalon szintén a görgővel/érintőpaddal lehet le és fel navigálni.

Minden menü hátterét az égitestek textúráját tartalmazó csomagból szereztem. A Main Menu, a Cards és a User Guide háttere valójában egy elsötétített verziója a Tejútrendeszernek, míg a szimuláció közben rendesen látszódik.

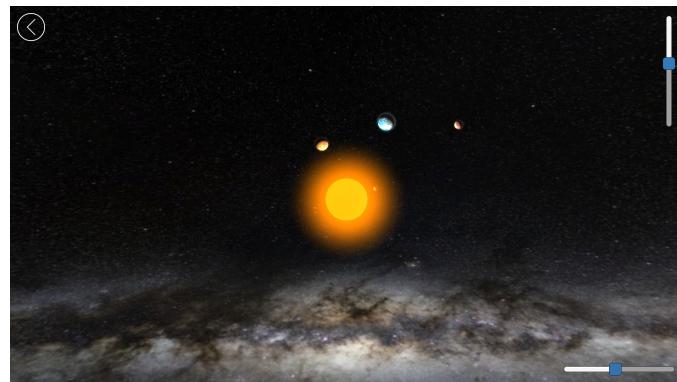


5. ábra. A szimulációs menü tartalma

A főmenüben a "START" gomb megnyomásával elindul a szimuláció (lásd: 5. ábra). Itt a görgő, valamint az érintő pad más funkcióval rendelkezik, mint az előző oldalakon a szimulációban a kamerát lehet közelebb vagy távolabb zoomolni vele, a középen lévő égitesttől. A szimulációs oldalon az égitesteken kívül egy, a fő menübe visszavezető gomb, illetve 2 csúszka található: a lépésköz és a perspektíva váltó csúszka.

IV.4.5. A lépésköz és a perspektíva váltó csúszkák

A lépésköz és a perspektíva váltó csúszkák csak a szimulációs ablakban megjelenő elemek. Nevüket direkt úgy választottam meg, hogy azok beszédesek legyenek.



6. ábra. Így néz ki, ha mindenktől csúszka el van mozgatva az alap állapotuktól

A lépésköz csúszka a jobb alsó sarokban található és feladata, hogy a szimulációban használt dt -t változtassa. Ennek a csúszkának a változtatása valójában a pontosságot változtatja (minél kisebb, annál pontosabb), de a felhasználó számára inkább sebesség változtatónak tűnik, hisz minél kisebb távolságot számol ki, annál pontosabb lesz, viszont több számolást is igényel a szimuláció. Ebből az okból kifolyólag nem ajánlom, hogy hosszútávon alacsony dt -vel fusson a program.

A bolygók szimulációjánál a maximum 5000, a minimum érték 1. A szimuláció kezdésekor a legmagasabb dt -vel indul el. Ezekkel a számokkal egész stabil eredményt nyújt.

A holdak szimulációjánál a legnagyobb dt 100, a legalacsonyabb 1. A szimuláció kezdetekor legtöbb esetben $dt = 50$, kivéve a Marsnál, ott 1. Sajnos itt nem sikerült egységes számot találni, ami stabil eredményt nyújt. A Mars holdjai magas dt esetén parittha módon kilőnek.

A perspektíva váltó csúszka célja, hogy más szögből figyelhessük meg az égitestek keringését. Ezt a Unity SetLocalPositionAndRotation és Lerp beépített függvényeinek segítségével sikerült megoldani. A SetLocalPositionAndRotation a lokális (a szülő objektumhoz viszonyított) pozíóját és forgását állítja be. Ez a függvény a pozíciót vektorként, a forgást kvaternióként várja el. A Lerp, a lineáris interpoláció függvény két pont közötti lineáris interpolációt adja meg. 3 paramétere a kezdőpont (a), a végpont (b) és a keresett pont helyzete, az intervallum hosszához képest (t , ahol $t \in [0, 1]$).

$$a + (b - a)t \quad (\text{IV.4.5.1})$$

A kezdőpozícióink a $(0, 0, -10)$ és a kezdőforgás $(0, 0, 0)$, a végpontunk a $(0, -10, 0)$ és a végforgás $(-90, 0, 0)$. A forgásokat Euler-szögekként adtam meg és kihasználtam Quaternion.Euler beépített függvényt, ami átváltja az Euler-szögeket kvaterniókká.[13]

IV.4.6. A zoomolás

A zoomolás csak a szimulációs menüben működő mechanika; a görgő segítségével lehet közelebbről vagy távolabbról megtekinteni a fókuszban lévő testet. A főkamera ortografikus módban van, mert így lehet változtatni a méretén, ezzel elérve egy zoom hatást. Ezt a hatást Unity SmoothDamp függvényével értem el, ami a görgő pozíciójának bekérésének, a megadott sebességnek, szorzónak és határoknak megfelelően változtatja a kamera ortografikus méretét (orthographicSize).[21]

IV.4.7. A fókuszálás mechanika

Az égitestek közül csak a bolygókra lehet fókuszálni, illetve alap állapotban a Napra fókuszál a kamera. Ha egyszer rákattint a felhasználó egy "Planet" címkevel ellátott bolygó objektumra, akkor az az objektum megkapja gyerek objektumként a kamerát. Így nem kell külön megírni a bolygó lekövetését megvalósító

szkriptet, mert a gyerek objektum minden követi a szülő objektumát, emiatt működik a perspektíva váltás is. Ennek a funkciónak van egy hibája, miszerint nehézen érzékeli az input-ot. Hipotézisem a probléma okozójáról az, hogy mivel a textúra valójában egy minta objektumhoz van kötve, ami több gyerek objektumból épül fel, így ez okozhat esetleg problémát a kattintás érzékelésben. Habár erről a hibáról tudok, megoldással sajnos nem sikerült előállnom.

Fókuszáláskor 2 dolog történik:

1. A Scene-en lévő összes bolygó, a kiválasztottan kívül, és a Nap objektumok megsemmisülnek.
2. Lefut a LoadMoonsOfPlanet metódus, ami betölti a bolygó holdjait.

Azért van szükség a megsemmisítésre, mert ahogy azt már korábban is kifejtettem (lásd: IV.4.2), más a távolságok skálázása, így enélkül összeütköznek. Ha a felhasználó másodjára rákattint a bolygóra, akkor a bolygó és a holdjai megsemmisülnek, majd újra legenerálódik a Nap és a 8 bolygó, illetve a kamera megint, szülő nélkül, a Napra fókuszál.

IV.4.8. Az információs kártyák

A szimuláció látványosan próbálja bemutatni az égitestek keringését, viszont ha valakit érdekelnek az adatok, akkor neki ajánlott a második menüt megnéznie. A Fő menüből elérhető a "CARDS" gomb megnyomása után az információs kártyákat tartalmazó menü (lásd: 4).

Ezt úgy szerettem volna megcsinálni, hogy informatív és játékos legyen. A kártyák dizájnjának megtervezéséhez, szerepjátékokban használt kártyákat vettettem alapul, miközben próbáltam a minimalista stílusra törekedni. Példának vegyük a Föld kártyát:



7. ábra. minden kártya ugyanazt a sablont követi.

Mint ahogy az látható, a kártya tetején, a fejlécben található a név, alatta egy választóvonallal elválasztva a kártya törzsében különböző adatok. Az adatok ugyanazok, mint amiket a szimulációban használtam, azaz szintén a *JSON* fájl-ból lettek kiolasva (bővebben lásd: IV.3). Szűréskor csak a *CelestialBodies* osztály adott listáján iterálunk végig, minden egyes alkalommal példányosítva egy kártya objektumot. Ha a felhasználó az összes kártyát szeretné látni, akkor az "All"-t kell kiválasztania, ilyenkor minden lista eleme megjelennek. A kártyákon lévő szöveg egy formátált sztring, a valós adatokat tartalmazza mértékegységekkel. Ennél több adatot is tartalmazott az adatbázis, viszont én ezeket tartottam a legfontosabbnak. Pár holdnál hiányzik a térfogat értéke, olyankor azt kihagyja.

IV.5. Javítási és továbbfejlesztési javaslatok

Habár a kódot én írtam, s meg vagyok elégedve a program jelenlegi állapotával, nem álltam magam azzal, hogy tökéletes lenne. Ebben a fejezetben pár javítási és továbbfejlesztési javaslattal szeretnék előállni.

IV.5.1. Javítandó hibák

A bolygókra való kattintás nem valami könnyű és inkonzisztens. Megoldást nem sikerült erre találnom, úgy tűnik, hogy a *Collider*-nek, ami a bolygó objektumokon van, bezavar a textúra.

A méretarányosságon is lehetne még mit dolgozni. A bolygóknál ez még nem is olyan nagy probléma, hisz nem ütköznek egymásba. Viszont, a holdaknál sajnos előfordul, mert egységes méretet állítottam be nekik, holott merőben különböznek egymástól. Ez a szimuláció szempontjából lényegében egy elhanyagolható részlet, hisz pontszerűnek tekintjük az égitesteket.

A holdak különösen problémásak, van amelyik nagyon szépen mozog és van, amelyik meghaladván a szökési sebességet kilő a semmibe. Ennek főleg a skálázás és a rosszul megválasztott, egységes lépésköz az oka. Mivel főleg a bolygók szimulációjára tettem a hangsúlyt, így nem maradt időm optimalizálni a holdak szimulációját.

IV.5.2. Továbbfejlesztési javaslatok, ötletek

Első továbbfejlesztési javaslatom a kártyákhoz megtekintés gombot rakni, ami megmutatja az adott égitestet a szimulációban. Ez összekötne a program két felét, amik jelenleg elégé elszeparáltak.

Második ötleteim, a program többnyelvűtétele. Ez fontos lenne, mert persillanat minden angolul van, a használati útmutatón kívül. Ez megnehezítené, hogy fiatalabbak használják tanuláshoz, mert sokan közülük nem tudnak még angolul.

Harmadik javaslatom: megjeleníteni az égitestek nevét az objektumokon. Ez szintén egy fontos kiegészítés lenne, mert habár a bolygókat szerintem mindenki meg tudja nevezni, a Szaturnusz 70 megjelenő, teljesen fehér holdját kétlem.

Negyedik, a bolygók forgását implementálni. Habár ezt terveztem lefejleszteni, sajnos a skálázás miatt ez perpillanat nem valami egyszerű. Ennek a leprogramozásához újra kell dolgozni a program azon részét is.

Ezeket idővel magam is implementálni tudtam volna, viszont a rendelkezésre álló időbe már sajnos nem fért bele. A következőkben olyan továbbfejlesztési javaslatok jönnek, amikhez vagy én egyedül kevés vagyok vagy, amiket nem is volt célom elsősorban belerakni a projektbe, de érdekes és hasznos lenne megcsinálni.

Az eredeti adatbázis, amit én leszűrve és letisztítva felhasználtam tartalmazott több információt, illetve több égitestet is, amit meg lehetne jeleníteni mind a szimulációban, mind a kártyák között. Sőt, akár más csillagokkal és azok bolygórendszerével is fel lehetne tölteni az adatbázist. Egy ilyen irányú esetleges bővítésre részlegesen fel lett készítve a kód. A mesterséges égitestek vagy űrrakéták kilövésének szimulációja is egy érdekes adalék lenne a programba.

Az általam felhasznált textúra csomag nagyon kevés égitest textúráját tartalmazta. Egy fontos fejlesztés lehetne ezeknek kibővítése, viszont én személy szerint nem értek a 3D modellező programokhoz.

A JSON fájl nem egy rossz módja az adatok tárolásának, de ha ezt bővíteni szeretnénk, akkor azt minden gépen, amin rajta van a program, külön meg kéne csinálni. Persze, ezt egy javítócsomag (ismertebb nevén *patch*) elterjesztésével is orvosolni lehet, viszont mennyivel egyszerűbb lenne, ha leváltanánk ezt egy *API*-ra. Akkor csak az adatbázist kell bővíteni és mindenkinél megtörténik a változás, amíg rendelkeznek internethozzáféréssel. Vagy lehetne akár mindenkitől egyszerre használni. Így biztosítva azt, hogy internet nélkül is működjön a program, de, ha rendelkezik kapcsolattal, akkor lementse az új adatokat.

V. Összegzés

A tézis végéhez közeledünk. A dolgozat folyamán bemutattam az általam elkezített szimulációs programot és a felhasznált technológiákat, valamint a fizikai törvényeket és matematikai képleteket, amik működtetik ezt a szimulációt. A projekt elkezdésekor kitűztem magamnak 4 célt:

1. A programnak látványosnak kell lennie, hogy a fiatalokat érdekelje, illetve 3 dimenziósnak, hogy különböző szemszögekből lehessen megfigyelni a égitestek keringését.
2. A szimuláció tartalmazza a Napot, a Naprendszer bolygóit és azok holdjait.
3. Az égitestek keringése legyen realisztikus.
4. A méretek legyenek arányosak.

Az első célomat véleményem szerint sikerült elérnem. A program háromdimenziós gömböket használ a keringésben, amiket különböző szemszögből lehet megtekinteni, felgyorsítani és lelassítani, valamint a fontosabb égitestek látványos textúrákkal lettek ellátva.

A felhasznált adatbázis szuboptimális léte miatt egyes holdak kimaradtak. Sajnos párnál hiányzott a tömeg értéke, ami nélkül nem működik a szimuláció. Viszont a Nap, az összes bolygó és azok legtöbb holdja megjelenik a szimulációban, így én a második célt sikeresnek tekintem.

A harmadik ponttal kapcsolatban csak részleges sikereket tudok felmutatni. Mint arra korábban kitértem, a hangsúlyt a bolygók keringésére fektettem, elhangyolván a holdak keringését.

A méretek arányosságát kudarcnak tekintem. Habár sikerült részleges sikereket elérni, hisz a távolságok arányosak lettek, az égitestek méretére nem sikerült arányt találni. Ez ugyebár nem probléma a szimuláció szemszögéből, mert az égitestek pontszerűnek tekintjük, viszont nem valami realisztikus.

Habár nem készítettem kimutatást arról, hogy a fiatalabb generációkat érdekli-e a program vagy emiatt jobban érdekelné a csillagászat, viszont a 9 éves húgom a játék kipróbálása óta határozottan magasabb affinitást mutat a téma iránt. Emiatt számomra már megérte megcsinálni.

Köszönetet szeretném mondani a konzulensemnek, Kiss-Vincze Tamásnak, a szakdolgozatomban való segítségéért! Köszönöm Dr. Király Balázs Gábornak, hogy átnézte a matematikai és fizikai részt! Szeretném köszönötetet mondani családomnak, akik mindenben támogattak engem! Továbbá köszönöm a segítőkész barátaimnak, hogy ha elakadtam a projektemben vagy a kiégés szélére kerülttem, mindig ott voltak számomra! Végül mindenkinél köszönöm, aki elolvasta ezt a dolgozatot!

Szakirodalom jegyzék

- [1] Richard P. Feynman, Robert B. Leighton és Matthew Sands. *A Feynman-előadások fizikából*. I. rész. Typotex, 2017. URL: <https://www.feynmanlectures.caltech.edu/>.
- [2] Érdi Bálint. *Égi Mechanika*. Tankönyvkiadó, 1989.
- [3] Robert S. Westman. *Nicolaus Copernicus*. URL: <https://www.britannica.com/biography/Nicolaus-Copernicus>.
- [4] Samuel J. Ling, William Moebs és Jeff Sanny. *University Physics*. 1. rész. URL: <https://pressbooks.online.ucf.edu/osuniversityphysics/>.
- [5] Ponori Thewrewk Aurél és tsai. *Csillagászati kisenciklopédia*. Gondolat Kiadó, 1969.
- [6] Robert S. Westman. *Johannes Kepler*. URL: <https://www.britannica.com/biography/Johannes-Kepler>.
- [7] Balázs Béla és tsai. *Csillagászat*. Akadémia, 1989.
- [8] Robert S. Westman. *Isaac Newton*. URL: <https://www.britannica.com/biography/Isaac-Newton>.
- [9] Dringó László. *Numerikus analízis II*. 12. kiad. Nemzeti tankönyvkiadó, 1994.
- [10] J. J. O'Connor és E. F. Robertson. *Carl David Tolmé Runge*. URL: <https://mathshistory.st-andrews.ac.uk/Biographies/Runge/>.
- [11] MARTIN WILHELM KUTTA. URL: <https://mathforcollege.com/nm/anecdotes/kutta.html>.
- [12] Anthony Ralston. *Bevezetés a numerikus analízisbe*. Műszaki Könyvkiadó, 1969.
- [13] Unity Documentaion. URL: <https://docs.unity3d.com/Manual/index.html>.
- [14] Inno Setup Documentation. URL: https://documentation.help/Inno-Setup/topic_whatisinnosetup.htm.
- [15] Paris Buttfield-Addison, Jonathon Manning és Tim Nugent. *Unity Game Development Cookbook: Essentials for Every Game*. O'Reilly Media, 2019.
- [16] Jamie Chan. *Learn C# in One Day and Learn It Well: C# for Beginners with Hands-on Project: Volume 3 (Learn Coding Fast with Hands-On Project)*. CreateSpace Independent Publishing Platform, 2015.
- [17] John Paul Mueller. *C# 10.0 All-in-One For Dummies*. For Dummies, 2022.
- [18] Robert Nystrom. *Game Programming Patterns*. 2014.

- [19] M. Fowler és K. Scott. *UML Distilled – A Brief Guide to the Standard Object Modeling Language (Object Technology Series)*. 3. kiad. Addison-Wesley, 2003.
- [20] Jon Skeet. *C# in Depth*. 4. kiad. Manning, 2019.
- [21] Anthony Davis, Travis Baptiste és Russell Craig. *Unity 3D Game Development: Designed for passionate game developers Engineered to build professional games*. Packt Publishing, 2022.
- [22] Alex Okita. *Learning C# Programming with Unity 3D*. Routledge, 2014.

NYILATKOZAT
az írásom eredetiségéről
(PTE SZMSZ 5. sz. mellékletének 14/1. számú melléklete alapján)

Alulírott Farkas Patrik Márk (név), BYONIZ (NEPTUN kód), büntetőjogi felelőségem tudatában kijelentem, hogy

Naprendszer szimulációs program Unity-ben

című írásomban foglaltak saját, önálló munkám eredményei, ennek elkészítéséhez kizárolag a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel, írásomat a Pécsi Tudományegyetem vonatkozó szabályzatainak betartásával készítettem. Tudomásul veszem, hogy a szerzői jogi szabályok betartását a Pécsi Tudományegyetem plágiumkereső rendszeren keresztül ellenőrizheti.

Pécs, 2024. Május 02.


(hallgató aláírása)