SZEGEDI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI ÉS INFORMATIKAI KAR
FIZIKAI INTÉZET

Diplomamunka

Bolygórendszerek paraméter- és lakhatóságizóna-evolúciójának vizsgálata Python-szimulációk segítségével

Studying the evolution of planetary system parameters and habitable zones using Python simulations

Murvai Adrián Csaba
Csillagász MSc

Témavezető:
Dr. Szalai Tamás
tudományos munkatárs

Szeged
2021

Tartalomjegyzék

1. Bevezetés **2**

2. Célkitűzés **6**

3. A Virtual Planet környezet **7**
3.1. Az **ATMESC** modul 8
3.2. Az **EQTIDE** modul 10

4. Példaszimulációk próbafuttatása **10**
4.1. A környezet inicializálása 11
4.2. Bemeneti fájlok 11
4.3. Kimeneti fájlok 13
4.4. Ábrázoló fájlok 15
4.5. A kiválasztott szimulációk 16
4.5.1. "Tidal Heating of Io" 17
4.5.2. "(Pre-)Main Sequence Stellar Evolution" 18
4.5.3. "Solar System Orbital Dynamics from Secular Theory" 23

5. Saját kódjaim fejlesztése **25**
5.1. Eredeti környezet 25
5.2. Grafikus kezelőfelület 29
5.3. A GUI felállítása 30
5.4. A EHZ mód felújítása 37
5.5. A **VPlanet**-tel nyert eredmények hasznosítása 39

6. Összefoglalás és kitekintés **40**

7. Köszönetnyilvánítás **40**

8. Függelék **41**

1. Bevezetés

Fizika BSc szakos projektmunkámban és szakdolgozatomban már beszámoltam a csillagászat egyes ágai iránt tanusított érdeklődésemről. Régóta foglalkoztat az exobolygók, a kolonizálás, valamint az idegen élet koncepciója. Még mindig nyitott kérdés, hogy hol is érdemes kutatni földönkívüli lakható világok, netán intelligens létformák után. Emiatt alkották meg a lakhatósági zónák definícióját, melynek a legegyszerűbb megfogalmazása: az a tartomány egy adott rendszer csillaga körül, melyen belül a víz folyékony halmazállapotban is előfordulhat. Természetesen ennél jóval komplikáltabb faktorokat is figyelembe tudunk, illetve figyelembe kell venni. Projektmunkám remek felvezetőként szolgált a szakdolgozatomhoz, utóbbi keretein belül pedig elkezdtem foglalkozni a Python környezetben működő programok fejlesztésével. A képzés végére elegendő tudást elsajátítottam, hogy az oktatásban és bemutatókon hasznos szoftvereket alkossak meg, ezáltal is növelve az programozásba vetett affinitásom. Az adott szkriptek működését alább részletezem.

Elsősorban fontosnak tartottam, hogy a legegyszerűbb modell előállítását elvégezzem. Ehhez természetesen szükség van némi matematikai háttérre is. Azt a közelítést alkalmazzuk ilyen számolásoknál, hogy az anyacsillag adott mennyiségű energiát sugároz környezetében, amit a szóban forgó planéta elnyel, azt pedig feketetestként sugározza vissza. Röviden azt mondjuk, hogy a bolygó által abszorbeált és emittált sugárzás mennyisége, energiája megegyezik. Ezen közelítésben figyelmen kívül hagyjuk a planéta esetleges atmoszféráját, ahogyan számos egyéb, hőháztartást befolyásoló folyamatot is. Alapul vesszük a - feketetest-sugárzónak tekintett - csillag energiakibocsátását, melyet az L luminozitás mennyiséggel jellemezhetünk:

$$L = 4\pi R^2 \sigma T_*^4 \tag{1}$$

Itt R a csillag sugara, T_* a csillag felszíni hőmérséklete, σ pedig a Stefan-Boltzmann állandó. Az adott távolsággal arányosan fog gyengülni ez az energiamennyiség, így könnyen eljutunk a fluxushoz:

$$S = \frac{L}{4\pi a^2} = \frac{4\pi R^2 \sigma T_*^4}{4\pi a^2} = \left(\frac{R}{a}\right)^2 \sigma T_*^4 \tag{2}$$

Itt S a napállandó, mely az adott felületelemre egységnyi idő alatt merőlegesen beeső energiát adja meg. A Föld esetében ez (a légkör tetején mérve) $S = 1367 \frac{W}{m^2}$.
A planéta annak A albedójától függő hányadot nyel el az adott sugárzásból, amit közelíthetünk úgy, hogy egy r bolygósugarú korongra érkezik. A bolygótest - az energiaegyensúly okán - kisugározza az elnyelt energiát, ami ismét leírható a Stefan-Boltzmann törvény segítségével, vagyis:

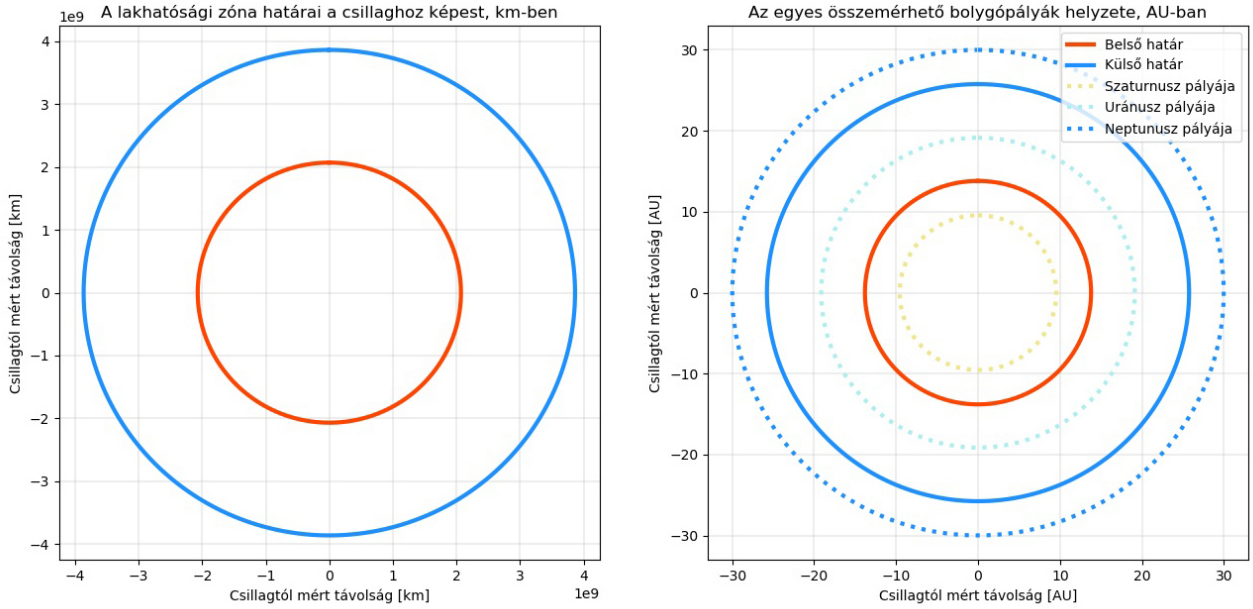
$$P_{abs} = S\pi r^2(1-A) = P_{em} = 4\pi r^2 \sigma T_e^4 \tag{3}$$

Itt P_{abs} és P_{em} rendre az abszorbeált és emittált energiamennyiségek. Megjelenik az egyenletben a keresett T_e egyensúlyi hőmérséklet, amelyre rendezve:

$$T_e = \left(\frac{S(1-A)}{4\sigma}\right)^{\frac{1}{4}} \tag{4}$$

Ha az utóbbi egyenletbe behelyettesítjük a napállandó képletét, rendezéssel megkaphatjuk az adott körülményekhez tartozó félnagytengely nagyságát. Ha az egyensúlyi hőmérséklet helyére 0 és 100 C° foknyi hőmérsékletet írunk K-ben kifejezve, akkor megkapjuk a klasszikus zóna határait.

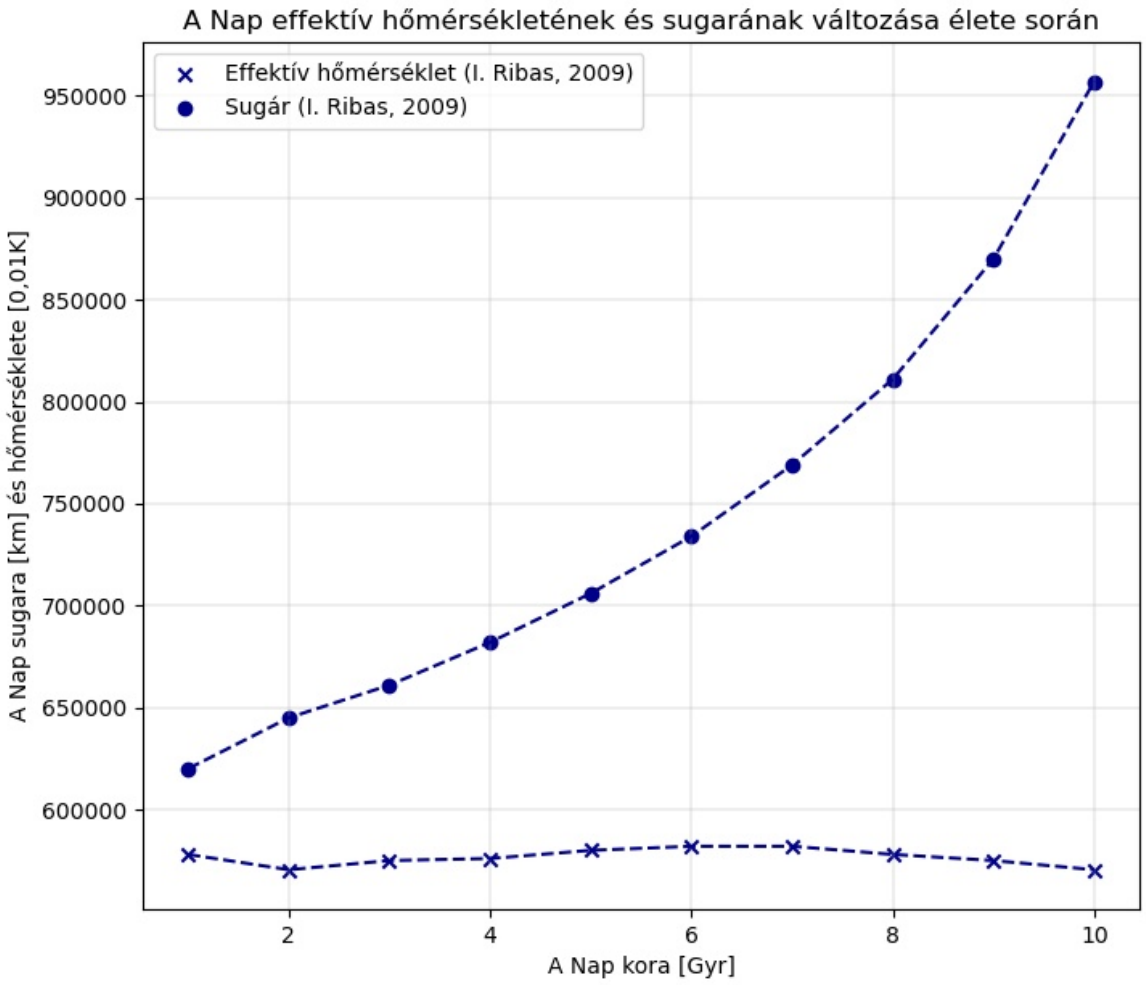
Első programomba három paramétert beírva eljuthatunk a legegyszerűbb értelemben vett lakhatósági zónához. Ezen megválasztandó értékek a centrális csillag sugara, hőmérséklete, valamint a kérdéses planéta albedója. Az eredményt a 1.1. ábra mutatja. Hogy szemléletesebb eredményt kapjunk, a kód ábrázolja a számolt zóna pozíciójával összemérhető néhány, Naprendszerünkben lévő bolygó pályájának helyzetét is.



1.1. ábra. Az első ábrázolószoftveremmel megjelenített ábrapár. A bal oldali panelen km mértékegységben láthatjuk az adott csillag körüli lakhatósági zóna határait. A jobb oldali ábrán ugyanezt látjuk AU-ban kifejezve, a zónával összevethető, naprendszerbeli, hozzávetőleges bolygópályákkal együtt.

Következő lépésként az időbeli fejlődés vizionálását próbáltam meg megvalósítani. Ehhez remek alapot szolgáltatott I. Ribas cikkének első ábrája, melyről adatsor hiányában szabadszemes becsléssel olvastam le az egyes sugár- és effektívhőmérséklet-értékeket saját csillagunk esetében [1]. Az erről készített 3. táblázat a függelékben található. Ezt követőleg sikeresen replikáltam az említett cikkben található grafikont, ami szintén jó feladatnak minősült az ábrakészítésre megalkotott *matplotlib* modul praktikáinak elmélyítésében. A rekreált grafikon a(z) 1.2. ábrán látható. A szoftver működése nagyon hasonló az előzőhöz, néhány funkcióbeli és esztétikai újítással együtt. Ebben az esetben már nem saját kézzel kell beírni az egyes adatpárokat, hanem egy beolvasni kívánt fájlt kell megadnunk. Tíz sorral dolgozik a program, futtatva tíz, 1.1. ábrához hasonló eredményt kapunk, külön-külön ábrázolva egy nagy panelen. A 3. táblázat adataival alkotott eredmény a(z) 1.3. és 1.4. ábrákon látható.

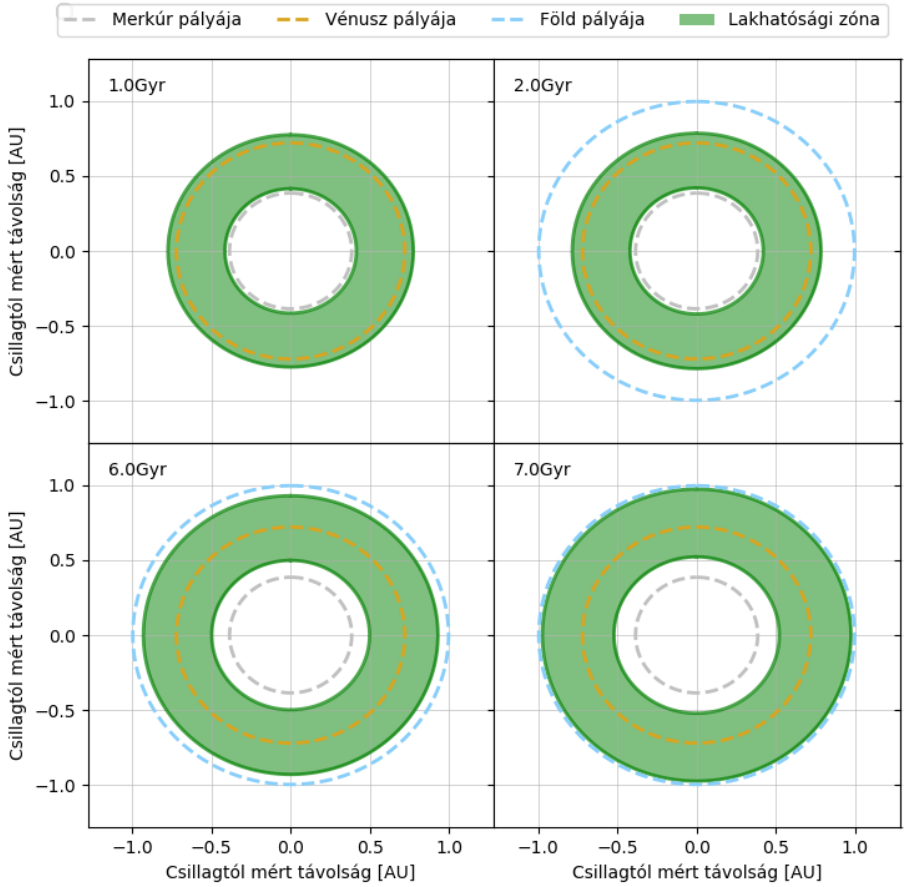
Az utóbbi szoftver továbbgondolása talán a legérdekesebb és leghasznosabb koncepció az egész témakörben - a folytonos lakhatósági zóna. Ez az a terület, amely attól is függ, hogy milyen időintervallumot tekintünk alapul. Adott idő alatt a planéta annak élete során fenntarthatja a rajta található vízkészletet, aminek szükséges feltétele, hogy mindvégig a zónában tartózkodjon és ne hagyja el azt. Míg a bolygó helyzete nagyjából állandó marad, addig a lakhatósági zóna a csillag fejlődésével és luminozitásának növekedésével folyamatosan tolódik kifelé. Napunk több, mint 5 milliárd év múlva földpálya sugarú gázgömbbé duzzad, ezzel az életnek kedvező sáv az óriásbolygókig, akár egészen a Neptunuszig is kitolódhat. Ezen elképzelés szoftveresen is egyszerűen megvalósítható. Alapul véve az evolúció-modellt, szük-



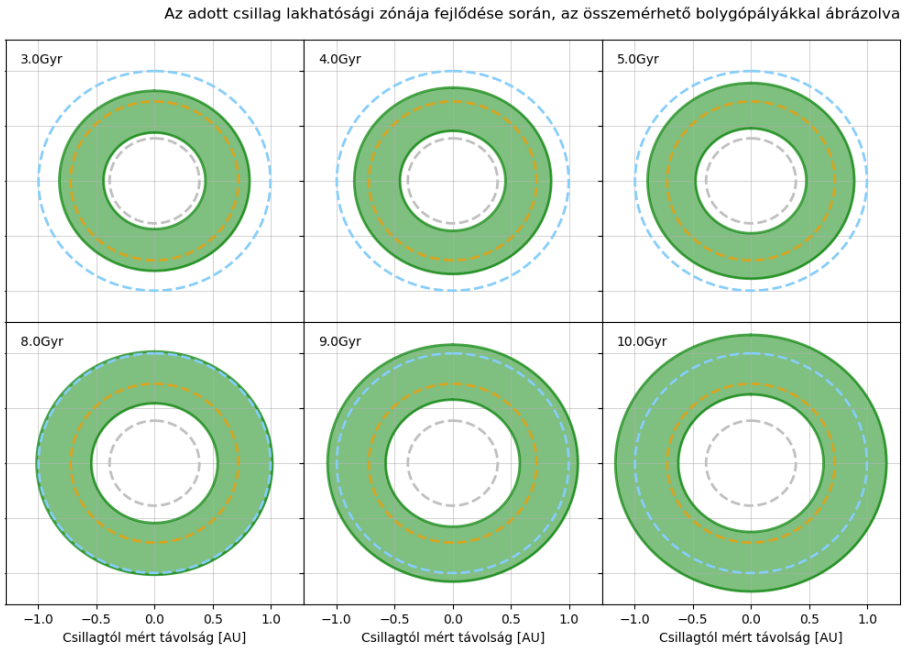
1.2. ábra. A Nap effektív hőmérsékletének és sugarának időbeli alakulása - saját készítésű ábra I. Ribas cikkének nyomán [1].

ségünk van két külön időpillanatban vett állapotra, melyekhez tartozik egy-egy sugár-effektív hőmérséklet adatpár. Ezeket felhasználva a szkript egymásra plottolja az adott gyűrűket, és azoknak a metszete fogja megadni a folytonos lakhatósági zónát. Ha az adott bolygó mindvégig a metszetben tartózkodik, akkor - egyéb fontos faktorokat még figyelmen kívül hagyva - nagy eséllyel fenn tudja tartani az élhető környezetet annak felszínén. Egy szemléletes eredményt láthatunk a 1.5. ábrán.

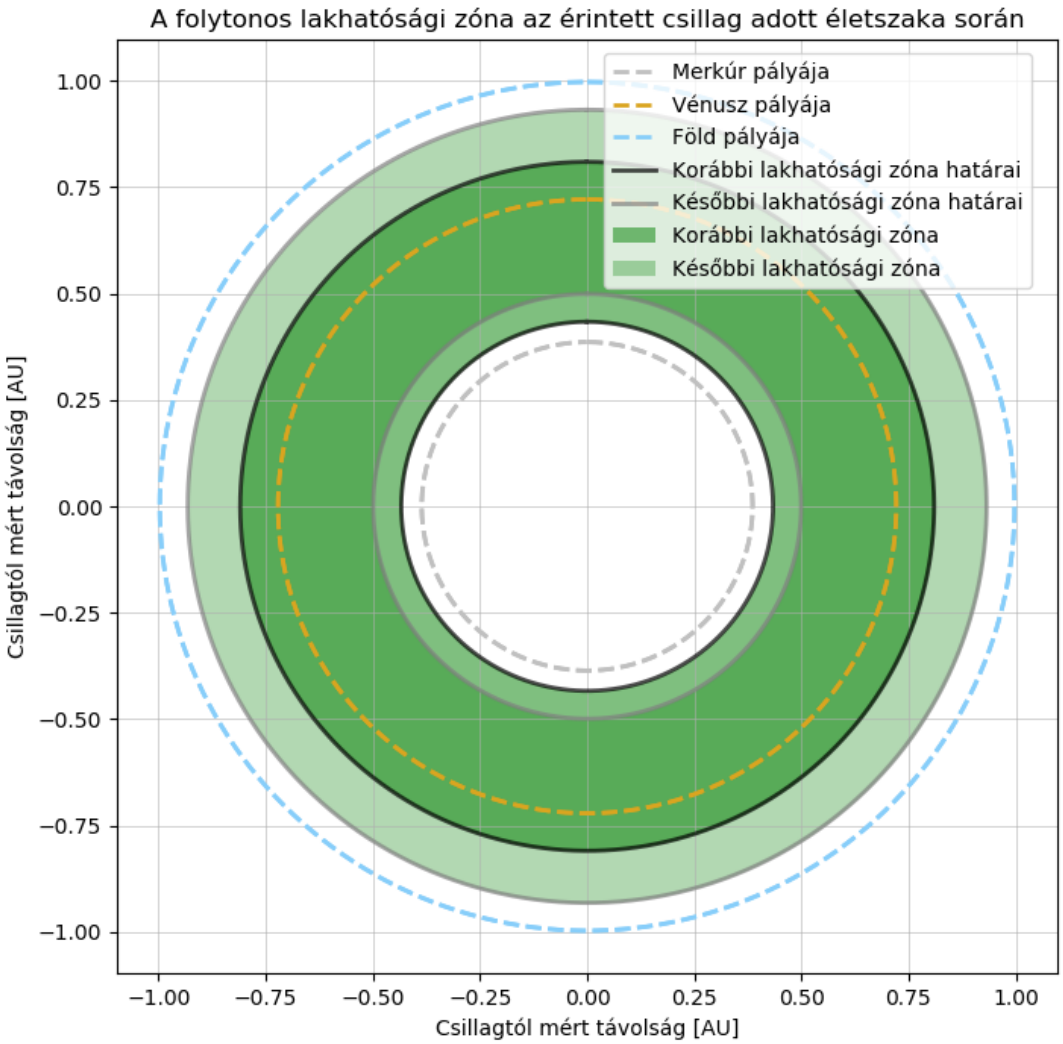
Ezen programok fejlesztése, felhasználása az MSc képzés keretein belül végezendő nyári szakmai gyakorlat, valamint diplomamunka egyik részét képezi. Munkám fő vezérvonala, hogy ebben a dolgozatban a lehetséges felhasználók számára beható leírást adjak az saját készítésű kódok kezeléséről és elvi működéséről, továbbá, hogy könnyen megérthető instrukciókkal lássam el az olvasókat az általam tesztelt és felhasznált szoftvercsomaggal kapcsolatban, amelyet a következő bekezdésekben részletezek.



1.3. ábra. A második ábrázolószoftveremmel készített lakhatósági zóna evolúciója - bal oldali panelek.



1.4. ábra. A második ábrázolószoftveremmel készített lakhatósági zóna evolúciója - jobb oldali panelek.



1.5. ábra. A program által kiszámolt és megjelenített zónák különbsége: a folytonos lakhatósági zóna.

2. Célkitűzés

A lakhatóság és lakhatósági zónák problémakörébe számos bonyolult faktort bele kell vennünk, hogy ne csak közelítő eredményhez jussunk, hanem azok tudományos relevanciával is rendelkezzenek. Habár eddigi kitekintéseimben csak a szoftvereim továbbfejlesztése állt, ehhez elengedhetetlen a megfelelő eszközök kézbevétele. Amerikai kutatók összefogásából létrejött egy sokoldalú programcsomag, a **VPlanet: The Virtual Planet Simulator** [2]. Ezen nyílt forráskódú szoftver számos modulja rengeteg problémát enged a felhasználóknak szimulálni. A kód alapja C-ben íródott, viszont egyes elemek Python környezetben is interpretálhatók, legfőképpen az ábrák. A szoftvernek megvan a maga nyelvezete is, melynek tanulmányozása szintén a munkám részét képezte, erről pedig részletes leírást is találhatunk a dolgozatom további bekezdéseiben. A **VPlanet** modulok előnye a tervezett nagyon rövid lefutási idő, számos konkurenciaszoftvert maguk mögött hagyva. Ezek mellett az egyes részek kombinálhatóak is, így bonyolultabb problémákat is könnyedén szimulálhatunk. A következő részekben az egyes modulok (kódjaimhoz mért) relevanciáját, azok használatát, valamint a kombinációjukból megalkotott problémák tanulmányozását és individuális esetek vizsgálatát is bemutatom.

3. A Virtual Planet környezet

Az említett modulcsalád olyan bolygó- és bolygórendszer-fejlődésbeli problémákat enged nekünk szimulálni, melyekben jelentős és figyelmen kívül nem hagyható folyamatok a

- belső szerkezetbeli,
- legköri,
- csillagbeli,
- pályabeli,
- galaktikus,
- stb. változások.

A szerzők által publikált cikkben nem csak az individuális szimulációk, hanem azok kombinálásával megalkotott problémák végeredményét is megtekinthetjük. Olyan jelenségeket reprodukáltak, amelyekhez (földi, napredszerbeli, valamint ismert csillagrendszerekben történt) megfigyelési eredmény is tartozik. Fő célja ezen moduloknak, hogy hozzávetőlegesen egyszerű, megérthető kódok segítségével tetszőleges rendszerekben történő folyamatokat tudjunk rekreálni, méghozzá valósághűen. Hatalmas előnye még, hogy figyelembe veszi a bolygófejlődést befolyásoló fizikai folyamatokat, visszacsatolást biztosítva így a felhasználónak. Remek jelölt egy ilyen szimulációsorozatra a TRAPPIST-1 rendszer. A tőlünk hozzávetőleg 40 fényévre található csillag egy M7,5 spektrálosztályú vörös törpe, nagyjából 2500 K felszíni hőmérséklettel [3]. Az ilyen gázóriásokhoz tartozó planéták (amennyiben vannak) nagyon közel keringenek anyacsillagukhoz. Ennek szemléltetésére felhasználtam az eddigi kódom, az eredmény a(z) 8.1. ábrán látható. A rendszerben számos folyamatot figyelembe kell venni a megfelelő modellezések érdekében. Ide tartoznak a mágnesesen rendkívül aktív vörös törpecsillag, a TRAPPIST-1a flerjelenségei, a csillag és a bolygók közti pályainterakciók, perturbációk, az árapályfűtés, a forgások fékeződése, valamint a légkörmennyiség elvesztése. Az említett modellekkel viszont mindezek szimulálhatóak.

Ezek ellenére sajnos nem lehetünk biztosak abban, hogy egy, a lakhatósági zónában keringő bolygó biztosan folyékony halmazállapotú víz hordozója. Ha figyelembe vesszük a Vénuszt, hozzá sok hasonló bolygó létezhet és létezik is, ugyanis az árapályfűtés könnyen vezethez "megszaladt" (az angol *runaway* terminológiából) üvegházhatáshoz. A modulok kombinálása számos ilyen problémára nyithatja fel a felhasználók szemét, akárcsak:

- a hőháztartásért, valamint a forgási és keringési fejlődésért felelős szkriptek kombinálásával egy-egy bolygó jégtakarójának (vagy akár teljes jégkorszakjainak) változását tudjuk nyomon követni;
- a csillagfejlődés és légkörvesztés szimulálására írt kódok keresztezésével jól visszakaphatjuk egy planéta vízkészletének változását.

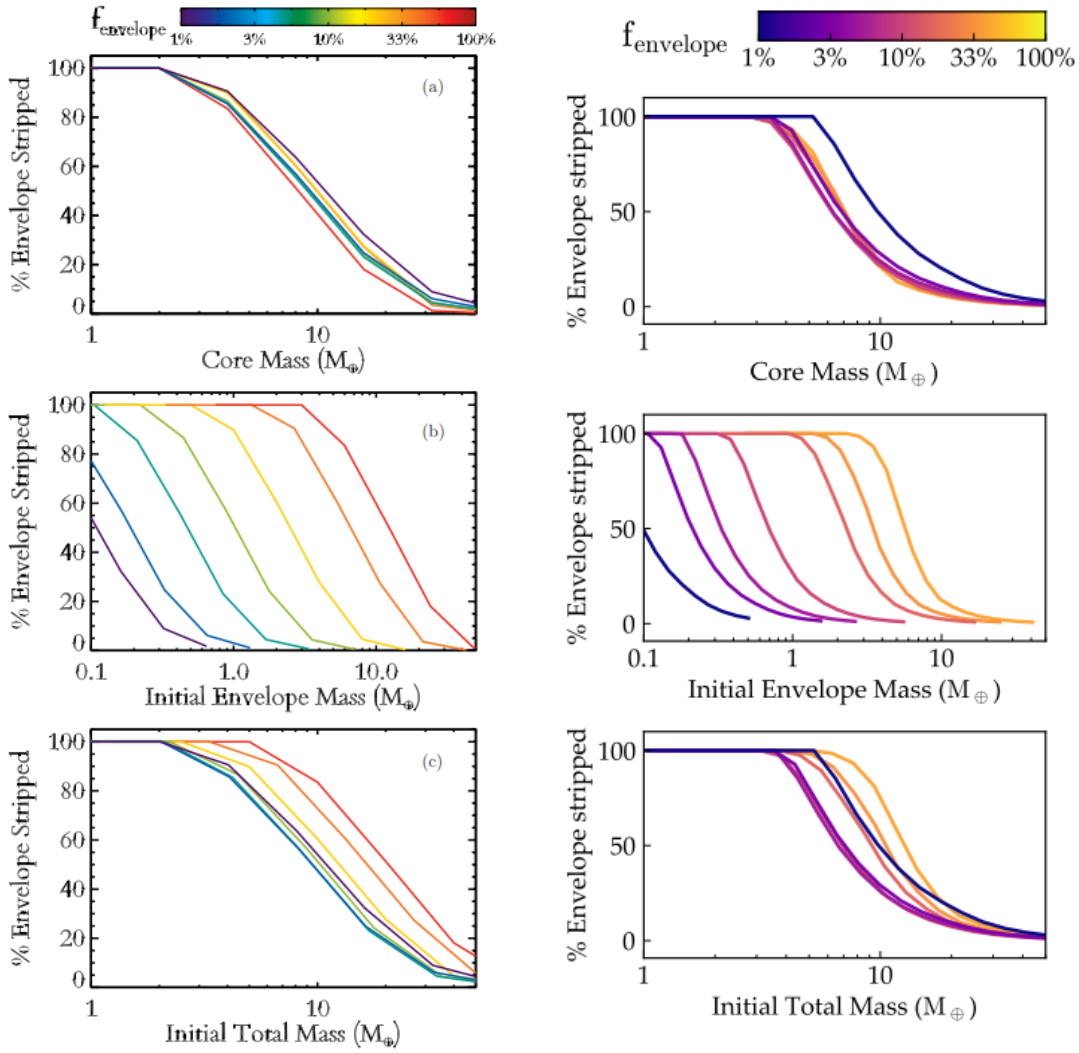
Mivel a **VPlanet** és a saját kódjaim jellege teljességgel eltér, így konkrét átfedést nem tudok elérni bennük. Míg az előbbi különféle problémák szimulációit futtatja és értékeli ki, addig az utóbbi zónákat kalkulál és ábrázol. Lehetséges modulbeépítésre nem látok módot, emiatt úgy döntöttem, hogy diplomamunkám keretein belül két szálon fogom végezni a kutatást és fejlesztést. Egyrészt tesztelni fogom ezt a viszonylag új és még kevesek által ismert programkörnyezetet, hogy valóban a várt eredményeket adja-e vissza. Amennyiben igen, érdekes esetekre is ki lehet terjeszteni a vizsgálódást, így akár új ötleteket is nyerve. Másrészt

mindenképp fontosnak tartom, hogy a kódírásban is magasabb szintre lépjek és olyan dolgokat legyek képes szimulálni, amik eddig nagy akadálynak tűntek. A kettő közötti keskeny átmenet pedig az ötletmerítés, hiszen számos érdekes és fontos faktorra rávilágít a **Virtual Planet**. Következzen néhány fontosabb modul, amelyek hitelt szerezhetnek az általam taglalt problémák körében.

3.1. Az ATMESC modul

Gyakran nem kielégítő feltétel, hogy egy bolygó a lakhatósági zónában keringjen, ugyanis az adott pozíció, habár legtöbbször szükségszerű, nem feltétlen implikálja a lakható környezetet, ugyanis számos egyéb tényező dönthet az egy adott planétán uralkodó klímaviszonyokról. Tekintsük például a Mars esetét. Kialakulása után néhány százmillió évvel hasonló körülmények uralkodhattak rajta, mint Földünkön. Lévén méretben feleakkora, tömegben pedig nagyjából tizede saját bolygónknak, nem rendelkezett megfelelően erős mágneses mezővel és gravitációval ahhoz, hogy kezdeti légtömegét megtarthassa, azt a napszélben érkező töltött részecskék és nagyenergiájú fotonok áradata egyszerűen lefújta. Légköri fejlődésben így számos dolog is szerephez jut. Pont ebben van segítségünkre az **ATMESC** (Atmospheric Escape) modul.

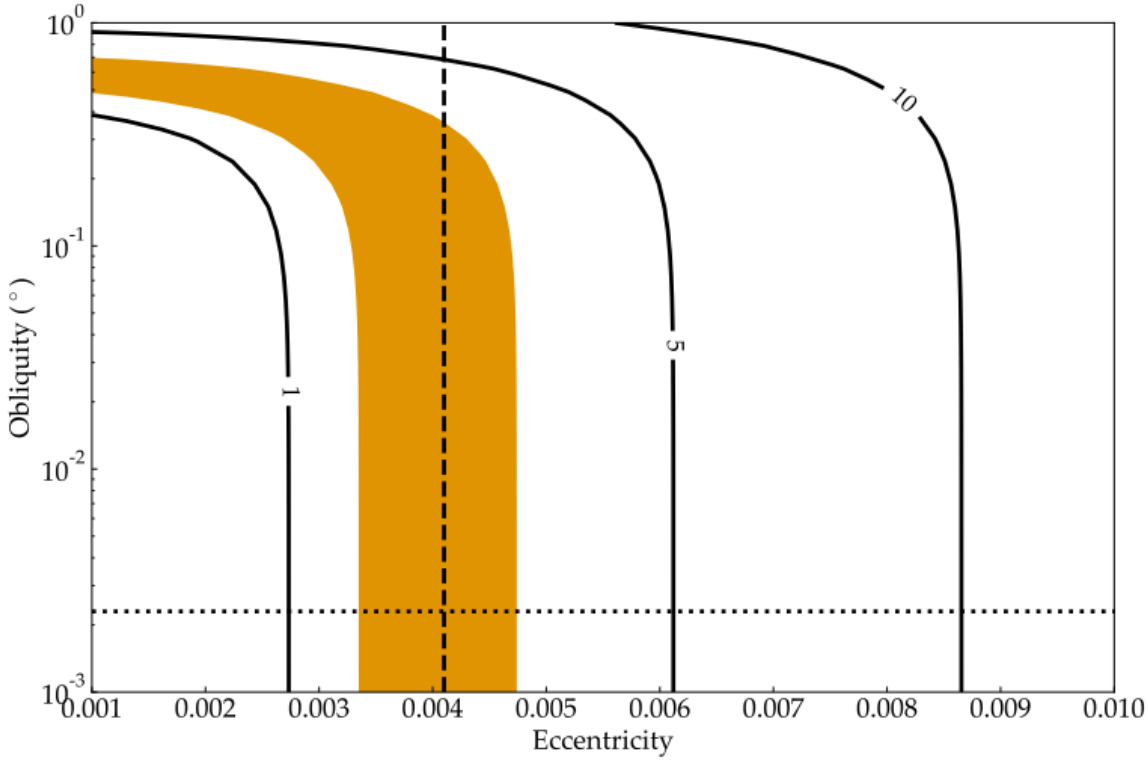
A programot bemutató cikkben a modul tesztelésére egy nem túl közeli rendszert választottak, a Kepler-36-ot. A csillag környezete két bolygónak ad otthont, melyből egyik egy minineptunusz, a másik pedig egy szuperföld, a két test sűrűsége pedig erősen eltér [4]. Ahogyan az utóbbiakra is igaz, legtöbb, csillagához közel keringő bolygó ki van téve a fotoevaporációnak. Hogy pedig miért ezt a rendszert választották, arra az a válasz, hogy össze tudták hasonlítani a kimenetelt egy korábban publikált eredménnyel [5]. A belső bolygót vizsgálták mindkét esetben, néhány megkötést pedig be kellett vezetniük a modell futtatásához. A planéta korát 5 milliárd évre becsülték, az általa elnyelt sugárzás pedig a földinek százszorosa. Futtatás után sikeresen visszakapták a már említett eredményt [5]. Ebből arra tudtak következtetni, hogy erős korreláció van a bolygó magjának tömege és az elvesztett atmoszféra mennyisége között. A két eredményt összehasonlításra alább feltüntetem a 3.1. ábrán. Egyik következő lépésnek érdemes választás lesz az adott modell futattása, így magam is megbizonyosodhatok a program hibátlan működéséről. Az erre tett kísérletről a 4. fejezetben olvashatunk.



3.1. ábra. A két kimenetel összehasonlítása. Bal oldalt Lopez & Fortney, jobb oldalt pedig a **VPlanet** eredménye [2][5]. Mindhárom panelpáron az elvesztett légtömeget láthatjuk, méghozzá a mag (felső), a légréteg (középső), valamint a bolygó (alsó) tömegének függvényében. A különböző színű görbékhez különböző kezdeti légréteghányad tartozik.

3.2. Az EQTIDE modul

Jelentős szerephez jut a lakható környezet kialakításában és fenntartásában (vagy akár elvesztésében is) az árapályfűtés, mely teljesen megváltoztathatja egy adott égitest hőháztartását. Az árapályerők ezen kívül hatással lehetnek még az egyes testek pályájára és forgására is. Ilyen hatást figyelhetünk meg a Naprendszerünk legdinamikusabban változó felszínű holdján, az Ión is. Az otthonát adó óriásbolygó, a Jupiter, valamint a hold paramétereit ismerve (tömeg, sugár, félnagytengely, excentricitás, tengelyferdeség) előállítható egy informatív grafikon, amelyen kijelölve láthatjuk azt a pontot, ami az adott árapályfűtés értékéhez tartozik. A mért árapályfűtés-fluxus $1,5-3\frac{W}{m^2}$, ez a tartomány pedig lefedi a szóban forgó értéket. Az elért eredmény a 3.2. ábrán látható. A fentebb részletezett példához hasonlóan ezt a szimulációt is megpróbáltam lefuttatni, az eredményekről szintén a 4. fejezetben olvashatunk.



3.2. ábra. Az Ión tapasztalható árapályfűtés mértéke (*surface tidal heat flux*) az excentricitás függvényében [2]. A (1, 5, 10) kontúrvonalak $\frac{W}{m^2}$ egységűek. A függőleges szaggatott vonal a hold megfigyelt excentricitása, míg a pontozott vízszintes vonal a becsült tengelyferdeség. A narancssárga régió a mért értékek által kijelölt tartományt mutatja. Láthatjuk, hogy a program visszaadja a helyesnek vélt eredményt.

4. Példaszimulációk próbafuttatása

Ahhoz, hogy a **VPlanet** használatát behatóbban megértsük, akár a matematikai/fizikai háttér ismerete nélkül, a kódok szintaktikáját részint át kell látnunk. Ezen feltételt kielégítve a következő fejezetekben levezetem, pontosan hogyan is tudjuk használni a szoftvert, valamint hogy miként épülnek fel a használni kívánt bemeneti, kimeneti, illetve egyéb fájlok, továbbá betekintést nyerünk majd a Python nyelvben megírt továbbfejlesztett ábrázolóprogramok jellegébe is.

4.1. A környezet inicializálása

A program igénybevételéhez szükséges környezet felállítása eltérő különböző operációs rendszereken. A következőkben saját tapasztalataimat fogom ismertetni, így a Windows 10 Pro verzión történő használat lesz előtérbe helyezve. Az egyéb rendszereken történő telepítésről szóló dokumentáció megtalálható a **VPlanet** hivatalos github oldalán [6].
Az instrukciókat végig szorosan követve próbálkoztam a szoftver telepítésével, ehhez először is egy emulátor terminált kellett installálni a számítógépre. Először a cygwin-nel próbálkoztam, ahogyan azt a leírás javasolta. A telepítésnél már eleve problémák adódtak, ugyanis a dokumentációban nincs részletezve, hogy pontosan mely csomagokat érdemes kiválasztani a listából, így sokadik alkalomra sikerült üzembe helyezni az emulátort. Ezelőtt természetesen egy igen pozitívnak vélt alternatívával, a MinGW-vel is megpróbálkoztam, sikertelenül. A telepítési problémát áthidalva újabb nehézségek elé néztem, így nem tudtam használni az említett programot. A hiba elhárítása saját tapasztalatok alapján sikeresen megtörtént, ugyanis a Windows operációs rendszerben alapértelmezett környezeti változókhoz manuálisan kellett hozzáadni a cygwin-hez vezető elérési útvonalat. Ez sajnos a dokumentációból hiányzó rész.

Az emulátor sikeres telepítése után rátérhettem a kívánt környezet felállítására, ehhez először is le kellett tölteni a teljes könyvtárat a nyíltan hozzáférhető github oldalról [7]. A megfelelő gcc parancs futtatása után az instrukciók alapján inicializáltuk a környezetet.

4.2. Bemeneti fájlok

A tipikusan előforduló fájlokat a dokumentációban is végigvezetett példa alapján fogom bemutatni. A vénuszi vízkészlet időfejlődése eléggé szemléletes, akárcsak bemenet és kimenet szempontjából is, így ezekről olvashatunk a következőkben.
A program *.in* bemeneti fájlokkal dolgozik, melyekből az egyik a főfájl (általában *vpl.in* néven), a többi pedig mellékfájl (a csillagok és bolygók paramétereiről). Az elsődleges, példában is *vpl.in* nevet viselő fájl tartalmazza a rendszerben található testeket és a szimulációra vonatkozó általános beállításokat, melyek jelen esetben a következők:

sSystemName	solarsystem	Rendszer neve
iVerbose	5	"Bőbeszédűség"
bOverWrite	1	Felülírható-e az esetleg meglévő fájl?
saBodyFiles	sun.in $ venus.in	Rendszerben lévő testek
sUnitMass	solar	Tömeg mértékegysége: g, kg, Föld, Neptunusz, Jupiter, Nap
sUnitLength	AU	Hossz mértékegysége: cm, m, km, Föld, Jupiter, Nap, AU
sUnitTime	YEARS	Idő mértékegysége: s, d, yr, Myr, Gyr
sUnitAngle	d	Szög mértékegysége: deg, rad
bDoLog	1	Készüljön-e log fájl?
iDigits	6	Maximum tizedesjegy
dMinValue	1e-10	Tengelyferdeség és excentricitás minimum egysége
dDoForward	1	Előre fejlődjön-e időben a rendszer?
dEta	0.01	Lépési időköz/felbontás
dStopTime	5e9	Evolúció vége
dOutputTime	1e6	Kimeneti időköz

Az egyes paraméterek első betűje fontos a bemenetekre vonatkozólag: b = Boolean (igaz/hamis), i = integer (egész szám), d = double precision (kettős pontosságú érték), s = string (szöveg), a = array (sor). Hasonlóan épülnek fel a csillag(ok) és a bolygó(k) paramétereit tartalmazó fájlok is. Jelen esetben a *sun.in* és a *venus.in* bemenetek részletezését láthatjuk alább:

sName	sun	Égiest neve
sModules	stellar	Használni kívánt modulok
saOutputOrder	Tim$e -LXUVStellar	Kimeneti értékek rendre
dMass	1.00	Csillag tömege naptömegben
dAge	5e7	Csillag kora a szimuláció elején
sStellarModel	baraffe	Evolúciós modell: "baraffe", none
sSatXUVFrac	1.e-3	Extrém UV luminozitás mennyisége
sSatXUVTime	1e8	Extrém UV luminozitás időskálája

sName	venus	Égiest neve
sModules	atmesc	Használni kívánt modulok
saOutputOrder	Time$ -LXUVStellar -SurfWaterMass$ -OxygenMantleMass	Kimeneti értékek rendre
dMass	-0.815	Bolygó tömege földtömegben
dRadius	-0.9499	Bolygó sugara földsugárban
dSemi	0.723	Félnagytengely
dEcc	0.006772	Excentricitás
dSurfWaterMass	-1.0	Kezdeti óceántömeg földi óceánmennyiségben
sWaterLossModel	lbexact	Vízkészletvesztési modell (Luger & Barnes (2015)
bInstantO2Sink	1	Oxigéngáz felszíni elnyelődése
sAtmXAbsEffH2OModel	bolmont16	UV abszorbciós modell

A fentebb látható, mínusz jellel ellátott paraméterek természetesen nem negatív értékeket jelölnek. Példaképp a Vénusznál láthatjuk, hogy *dMass*-nál -0,815 szerepel, ilyenkor ez felülírja a főfájlban található kért mértékegységet és földtömegben lesz megadva a bolygó tömege. A fájlok beparaméterezése után rátérhetünk a futtatásra, melyet a *vplanet vpl.in* paranccsal tehetünk meg. Ha a verbosity szintet, avagy a bőbeszédűséget a javasolt 5 értékre állítjuk, akkor a következőt láthatjuk a terminálban:

```
INFO: sUnitMass set in vpl.in, all bodies will use this unit.
INFO: sUnitTime set in vpl.in, all bodies will use this unit.
INFO: sUnitAngle set in vpl.in, all bodies will use this unit.
INFO: sUnitLength set in vpl.in, all bodies will use this unit.
INFO: sUnitTemp not set in file sun.in, defaulting to Kelvin.
INFO: sUnitTemp not set in file venus1.in, defaulting to kelvin.
INFO: sUnitTemp not set in file venus2.in, defaulting to kelvin.
INFO: sUnitTemp not set in file venus3.in, defaulting to kelvin.
INFO: dRotPeriod < 0 in file sun.in, units assumed to be Days.
INFO: dMass < 0 in file venus.in, units assumed to be Earth masses.
INFO: dSemi < 0 in file venus.in, units assumed to be AU.
```

```
INFO: dRadius < 0 in file venus.in, units assumed to be Earth radii.
INFO: dRotPeriod < 0 in file venus.in, units assumed to be Days.
INFO: dSurfWaterMass < 0 in file venus.in, units assumed to be
Terrestrial Oceans (TO).
INFO: dMinSurfWaterMass < 0 in file venus.in, units assumed to be
Terrestrial Oceans (TO).
INFO: dJeansTime not set for body venus, defaulting to 3.16e+16 seconds.
Input files read.
INFO: Age set in one file, all bodies will have this age.
INFO: sOutFile not set, defaulting to solarsystem.sun.forward.
INFO: sOutFile not set, defaulting to solarsystem.venus.forward.
INFO: sIntegrationMethod not set, defaulting to Runge-Kutta4.
INFO: dEnvelopeMass < dMinEnvelopeMass. No envelope evolution will be included.
INFO: dEnvelopeMass < dMinEnvelopeMass. No envelope evolution will be included.
INFO: dEnvelopeMass < dMinEnvelopeMass. No envelope evolution will be included.
INFO: Radius of Gyration set for body 0, but this value will be computed from
the grid.
All of sun's modules verified.
All of venus's modules verified.
Input files verified.
Log file written.
```

Az említett negatív érték funkciójáról az előbb látott sorok is megbizonyosodást adnak nekünk, példaképp:

```
INFO: dMass < 0 in file venus.in, units assumed to be Earth masses.
```

Természetesen, ha az adott verbosity értéket 0-ra állítjuk, akkor egy idő után egyszerűen csak visszakapjuk a terminálról a vezérlést és a szükséges fájlok el is készültek. Érdemes ámbár megtartani az alapbeállítást, ugyanis a program folyamatosan tájékoztat minket a szimulációk egyes lépéseiről, így, ha bármi problémát vétettünk, visszakövethetjük, hogy honnan ered pontosan a hiba oka. A verbosity értéktől függetlenül, a szimuláció lezárultával az alábbi sorokat láthatjuk majd. Innen tudhatjuk, hogy a futtatás véget ért és sikeresen zajlott:

```
Evolution completed.
Log file updated.
Simulation completed.
```

Ezután szemügyre vesszük a próbaszimulációból kapott kimeneti fájlokat.

4.3. Kimeneti fájlok

Több kimeneti fájlt találunk majd egy-egy eredményes futtatás után az adott könyvtárban. Minden esetben készül egy, az adott rendszer egészére jellemző logfájl, melynek a neve a bemeneti főfájlban meghatározott szöveget, vagyis a rendszer nevét fogja tartalmazni. Jelen esetben ez pontosan *solarsystem.log*, aminek egy rövidített változata alább látható:

```
Executable: vplanet
Version: <GITVERSION>
System Name: solarsystem
Primary Input File: vpl.in
Body File #1: sun.in
```

```
...
---- INITIAL SYSTEM PROPERTIES ----
(Age) System Age [sec]: 1.577880e+15
(Time) Simulation Time [sec]: 0.000000
...
----- BODY: sun ----
Active Modules: STELLAR
(Mass) Mass [kg]: 1.988416e+30
...
----- STELLAR PARAMETERS (sun)------
(LXUVStellar) Base X-ray/XUV Luminosity [LSUN]: 0.000677
Output Order: Time[year] LXUVStellar[LSUN]
----- BODY: venus ----
Active Modules: ATMESC
(Mass) Mass [kg]: 4.867332e+24
...
----- ATMESC PARAMETERS (venus)------
(SurfWaterMass) Surface Water Mass [TO]: 1.000000
...
Output Order: Time[year] SurfWaterMass[TO] OxygenMantleMass[bars]
---- FINAL SYSTEM PROPERTIES ----
(Age) System Age [sec]: 1.467428e+17
(Time) Simulation Time [sec]: 1.451650e+17
...
----- BODY: sun ----
Active Modules: STELLAR
(Mass) Mass [kg]: 1.988416e+30
...
----- STELLAR PARAMETERS (sun)------
(LXUVFrac) X-ray/XUV Luminosity Fraction []: 8.892684e-06
...
----- BODY: venus ----
Active Modules: ATMESC
(Mass) Mass [kg]: 4.867332e+24
----- ATMESC PARAMETERS (venus)------
(SurfWaterMass) Surface Water Mass [TO]: 0.000000
(OxygenMantleMass) Mass of Oxygen in Mantle [bars]: 199.365415
...
```

Ebben a fájlban minden érték SI mértékrendszerben van megadva, így biztosra mehetünk az értékek valósságáról. Emellett visszakövethetjük, hogy a kezdeti paramétereket biztosan helyesen adtuk-e meg, ugyanis mind azokat, mind a végső értékeket feltünteti a logfájl. Amit még itt láthatunk, az a kimenetekbe kért mennyiségek, vagyis az egyes oszlopok, továbbá a felhasznált bemeneti fájlok és modulok.

Ezek után vegyük sorra az egyéni, *.forward* kiterjesztésű kimeneti fájlokat. A fájlnevek felépítése először tartalmazza a már fentebb említett rendszernevet, valamint az adott égitest megnevezését. Így jelen esetben két ilyen fájlunk van, a *solarsystem.sun.forward* és a *solarsystem.venus.forward*. Ezekben annyi oszlopot találunk majd, ahány kulcsszót megadtunk a bemeneti fájlok *saOutputOrder* paraméterénél, továbbá a sorrend is ez alapján értendő. Tekintsünk meg egy-egy részletet a fájlokból, először a Napra vonatkozólag:

```
0.000000     0.000677
1.000000e+06 0.000677
2.000000e+06 0.000678
3.000000e+06 0.000678
4.000000e+06 0.000678
...
```

Emellett pedig a Vénuszra kapott eredményből egy részlet:

```
0.000000     1.000000 0.000000
1.000000e+06 0.978763 4.238860
2.000000e+06 0.957524 8.477719
3.000000e+06 0.936283 12.716579
4.000000e+06 0.915044 16.955439
...
```

Utóbbinál rendre az időt (Myr), a felszíni vízkészletet (mivel a bemenetnél negatív jellel láttuk el az értéket, így földi óceánmennyiség a mértékegység), valamint a köpeny által abszorbeált oxigénmennyiséget (szintén negatív jelet használtunk az input fájlnál, így bar mértékegységben kaptuk meg az eredményt) láthatjuk. Ha nem vagyunk biztosak a mértékegységekben, bármikor egyszerűen visszanézhetjük a logfájlt vagy a bemeneti fájlokat, hogy ellenőrizzük azokat.

4.4. Ábrázoló fájlok

Természetesen a szimuláció nem lenne teljes az adatok ábrázolása nélkül. A kapott eredményeket manuálisan is plottolhatnánk, de a fejlesztők erre külön-külön kódokat alkottak meg a legtöbb példánál. Ilyen intuícióval készült el a kimeneti fájlokkal konzisztens *vplot* modul, melyet egy Python szkriptbe importálva megkönnyíthetjük a saját dolgunkat ábrázolás terén. A könyvtárat az alábbi sorral tudjuk meghívni:

```
import vplot as vpl
```

Természetesen az "as vpl" csak rövidítés, enélkül is használható a könyvtár, viszont a fejlesztők mindenhol így hivatkoznak rá, így meghagyjuk ilyen formában. Mivel a tesztszimuláció egy leegyszerűsített, egybolygós változata a valóban elérhető esetnek, így abban nem lenne szükség az alább látható sorként kezelni az egyes bolygók adataiból nyert bemeneteket, ennek ellenére a kód így is működhet. A szkript további, érdemi része:

```
output = vpl.GetOutput()
time = output.sun.Age
planets = output.bodies[1:]
N = len(planets)
water = np.array([planet.SurfWaterMass for planet in planets])
oxygen = np.array([planet.OxygenMantleMass for planet in planets])
```

A fent látható *vpl.GetOutput()* függvény beolvassa a kimeneti *.forward* kiterjesztésű fájlokból az egyes oszlopokat és hozzájuk rendeli a paraméterfájlokban használatos mennyiségeket, így minden oszlopra az adott néven tudunk hivatkozni. Példaként a bolygó paraméterezésénél megadtuk, hogy szeretnénk kiíratni a felszíni vízmennyiséget és az elnyelt oxigént, ezekre a Python kódban is hasonlóképpen hivatkozhatunk, mint *SurfWaterMass* és *OxygenMantleMass*.

Hogy pontosan melyik paraméterre hogyan kereshetünk rá, vagy hogy melyek azok a bemeneti és kimeneti lehetőségek, amiket használhatunk, azt készítők hivatalos github linkjén találhatjuk meg, amit alább mellékeltem is [8].

4.5. A kiválasztott szimulációk

Az egyes tesztelendő példák kiválasztásánál számos szempontot figyelembe vettem. Látni fogjuk, hogy igyekeztem mind az egyszerűbb, mind az összetettebb problémák közül néhányat megvizsgálni, ami a futtatásokat illeti. Emellett nagyon fontos, követendő vonal volt, hogy megpróbáljak a saját programomba is olyan újdonságo(ka)t vinni, amik a **VPlanet** nélkül nem lennének kivitelezhetőek. A sikeres szimulációk mellett néhány sikertelen esetet is feltüntetek alább, körbejárva az esetleges technikai oldalát is a környezetnek, utalást adva arra vonatkozólag, hogy a felhasználók ilyen problémákba is ütközhetnek. A következő, 1. táblázatban sorra vesszük a kiválasztott szimulációkat, mellettük feltüntetve a felhasznált/felhasználandó modulokat, a hozzávetőleges lefutási időt, valamint a futtatás sikerességét. Rendszerességet is felfedezhetünk a sorokban, ugyanis fentről lefelé haladva a tesztek bonyolultsága és összetettsége a modulok számát illetően egyre növekszik. Emellett, mint láthatjuk, a legtöbb szimulációban az **ATMESC** és **STELLAR** modulok szerepelnek. Előbbi a bolygólégkör fejlődése miatt tölt be hasznos szerepet, utóbbi pedig a csillagok evolúciójáért felelős. Az utolsó két, elválasztott sornak külön szerep jut a saját programom szempontjából, erről a további alfejezetekben olvashatunk.

Szimuláció neve	Felhasznált modulok	Futási idő [perc]	Futtatás eredménye
Water Loss on Venus	ATMESC, STELLAR	1	Sikeres
Orbital Evolution of Circumbinary Planet Kepler-16b	BINARY	1	Sikeres
Radiogenic Heating of Earth	RADHEAT	<1	Sikeres
Tidal Locking of Gl-581d	EQTIDE	<1	Sikeres
Tidal Heating of Io	EQTIDE	-	Sikertelen
Kepler-36 Atmospheric Escape	ATMESC, STELLAR	<1	Sikertelen
Accumulation of Atmospheric Oxygen	ATMESC, STELLAR	<1	Sikertelen
Magma Ocean Evolution of Earth	ATMESC, MAGMOC, STELLAR	3	Sikeres
Evolution of a Planet Orbiting a Short-Period Binary	BINARY, EQTIDE, STELLAR	>180	Sikertelen
Magma Ocean Evolution of Planet TRAPPIST-1g	ATMESC, EQTIDE, MAGMOC, RADHEAT, STELLAR	5	Sikeres
(Pre-)Main Sequence Stellar Evolution	STELLAR	5	Sikeres
Solar System Orbital Dynamics from Secular Theory	DISTORB, DISTROT	5 >60	Sikeres Sikertelen

1. táblázat. Az elvégzett tesztfuttatások a hozzájuk tartozó modulokkal, a hozzávetőleges lefutási idővel, valamint az eredményükkel. Kihúzott számadattal azt a szimulációt jelölöm, amelynél meghiúsult a futtatás.

Az egyes, 1. táblázatban látható szimulációk meghiúsulásának több oka is volt. Az Ió árapályfűtésével kapcsolatos tesztről külön fejezetben írok, térjünk itt rá a másik esetre. Rögtön az említett probléma után látható két másik szimulációnál identikus a meghiúsulás oka.

Mindkét esetben a főfájl mellé még 20 másik fájlt is beolvas a program (*p00.in, p01.in...*). A lefutási idő igencsak gyors a megjósolt néhány perchez képest, ha azt vesszük alapul, hogy a legtöbb eredményre több időt kellett várnom, mint amennyit a készítők becsültek. Ez az eredményekben is látszik, a kérdéses adatsorok mindössze néhány sorból állnak. Emellett jelentős problémát okozott a *vplot* könyvtár helytelen működése, ami sajnos az összes esetet végigkísérte, viszont legtöbbször az ábrázolásért felelős kód - néhány sorral kiegészítve - minden gond nélkül működött, habár ez egyelőre kivitelezhetetlen nagyszámú kimeneti fájlok esetén.

4.5.1. "Tidal Heating of Io"

Ahogyan a cím sugallja, az Ió holdunkra ható árapályfűtést tárgyaló problémát vizsgálhatjuk az adott szimulációval. Erről néhány szót már olvashattunk a 3.2. fejezetben, emiatt tettem próbát a futtatásra. Ez a példa jelentősen eltér a többitől, ugyanis egy teljesen új könyvtárra/modulra is szükség van, amelyet a készítők *VSPACE*-nek neveztek el. Ennek a funkciónak a lényege, hogy bizonyos paraméterekre ne egyetlen érték legyen a bemenet, hanem ezzel egy nagyobb intervallumot fedjünk le a kért változókban. Ezt a könyvtárat alkalmazva kissé módosítanunk kell az input fájlokat, mellékelve egy *vspace.in* szövegfájlt is, melynek tartalmaznia kell a megfelelő elérési útvonalat, a kimeneti mappát, a vizsgálandó (és esetleg felülírandó) értékeket, valamint azok eloszlását. Egy példa erre:

```
srcfolder  .
destfolder data

file   vpl.in
file   jupiter.in
file   io.in

dEcc  [1e-3,0.01,n30] ecc
dObliquity [1e-3,1,l30] obl
```

Fentebb láthatjuk a bemeneti fájlokat, amiket a *VSPACE* modul figyelembe vesz, alattuk pedig a szóban forgó paramétereket. Az első része a sornak (pl. dEcc) az adott változó betűpontos elnevezése. Pontosan ugyanaz, mint ahogyan azt a bemeneti fájlokban és a már említett listában láthatjuk [8]. A harmadik rész (pl. ecc) a tetszőleges elnevezés, mely alapján az elkészült fájlokat megtalálhatjuk a létrejött alkönyvtárakban (példa egy kimeneti fájlra: *test/trial_ecc1obl1*. Végül a második rész (pl. [1e-3,0.01,n30]) szerepe a mintavételezés definiálása. A modul két lehetséges móddal rendelkezik: rács és véletlenszerű. Előbbihez - ha egzaktak akarunk lenni - megadhatjuk az adott paraméter kezdeti és végső értékét, illetve köztük a lépésközt. A bemenet lehet explicit, lineáris, vagy logaritmikus. Egyenletes, általunk megadott, 0,1 értékű osztásköz:

```
dSemi  [1,2,0.1] semi
```

Lineáris módban azt adhatjuk meg, hogy az intervallum hány részre legyen felosztva, alábbi példában 11:

```
dSemi  [1,2,n11] semi
```

Végül logaritmikus felosztásban is kérhetjük, a példában 1 és 1000 közt 10 értékre:

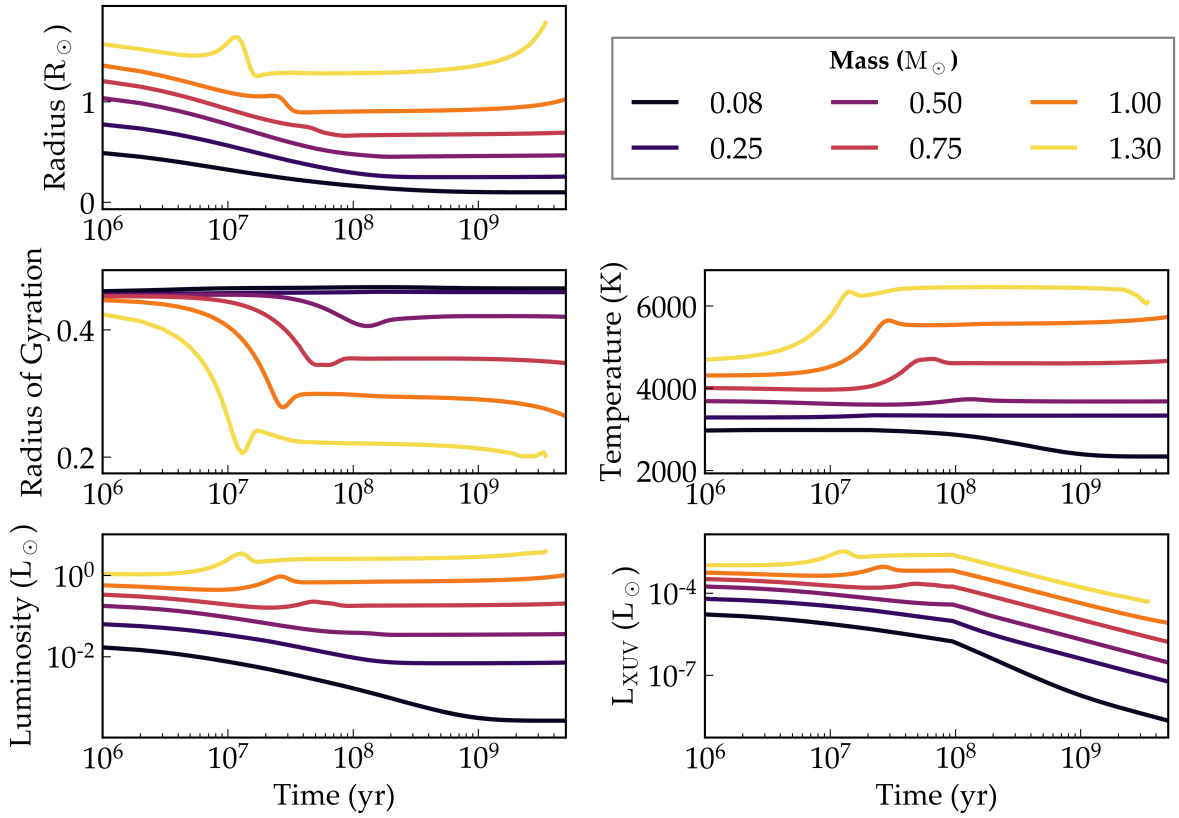
```
dSemi  [1,1000,l10] semi
```

Véletlenszerű értékmezőt is képezhetünk, négy almóddal, melyek "uniform", normális (Gauss-), illetve szinusz- és koszinusz-eloszlás. Rendre meg kell adni a következő értékpárokat: minimum és maximum érték, középérték és félértékszélesség, minimum és maximum szögérték; melléjük pedig a megfelelő kezdőbetűket kell mellékelni (u, g, s, c). Erről részletesebben az idetartozó dokumentációban is tudnak tájékozódni az olvasók. Ezek alapján láthatjuk, hogy a szóban forgó modul hasznossága és innovatív jellege figyelemre méltó.

Ezzel szemben figyelembe kell venni, hogy egy adott szoftvert nem lehet elégszer ellenőrizni és az egyes észrevételeket feljegyezni, ugyanis az instrukciókat követve a telepítés sikertelen volt, így sajnos a szimuláció tesztelését el kellett vetnem.

4.5.2. "(Pre-)Main Sequence Stellar Evolution"

Ezt a szimulációt a próbaprobléma elemzése után választottam, ugyanis úgy gondoltam, végre áthidalhatom a **VPlanet** és a saját programom közti szakadékot, vagyis előbbiből olyan használható információt nyerhetek ki, amit az utóbbiban kamatoztathatok. Maga a teszt problémamentesen működött az eredeti paraméterekre. Az erről készült, rekreált, eredetivel identikus grafikon a(z) 4.3. ábrán látható.



4.3. ábra. A **STELLAR** modult felhasználó szimuláció futtatása után sikeresen elkészített grafikonok. Látható, hogy az 1,3 naptömegű csillagnál a szimuláció hamarabb megáll, ez pedig egy specifikus kapcsolónak köszönhető a bemeneti fájlokban, ugyanis ezzel csak a fősorozati állapot végéig vizsgáljuk a fejlődést.

Az itt látható, összetettebb grafikon elkészítéséhez többszöri futtatás szükséges, jelen esetben hat különböző naptömegértékre. A szóban forgó szimulációnál, mivel egyetlen testet vizsgálunk, összesen két bemeneti fájl szükséges, tekintsük meg először a csillagra vonatkozót, mely az egyszerűség kedvéért a *star.in* nevet viseli:

```
sName                   star
saModules               stellar
dMass                   1
dAge                    1e9
sStellarModel           baraffe
bHaltEndBaraffeGrid     0
dSatXUVFrac             1.e-3
dSatXUVTime             -0.1
saOutputOrder           Time -Radius Temperature
```

A fent látható *bHaltEndBaraffeGrid* kapcsoló a "baraffe" csillagfejlődési modellre vonatkozik, maga a változó pedig azt befolyásolja, hogy az evolúció megálljon-e a fősorozat végén [9]. Az utána látható *dSatXUVFrac* paraméter azt szimbolizálja, hogy a csillag által kibocsátott bolometrikus luminozitáshoz képest mekkora az XUV (extrém ultraibolya) sugárzás hányada. Az érték 0 és 1 között kell, hogy legyen, értelemszerűen. Ilyen hullámhossztartományba eső sugárzást a csillag konstans mértékben bocsát ki az úgynevezett "szaturált" vagy "telített" fázisában. Az utolsó előtti paraméter, a *dSatXUVTime* pont ennek a fázisnak az időtartamát befolyásolja. Mivel a készítők az összes szimulációnál adott értéken hagyták a szóban forgó kapcsolókat, én is a megadott beállításokkal dolgoztam tovább. Fontos megjegyezni továbbá, hogy ebben a fájlban tudjuk módosítani az égitest kezdeti korát.

Vegyük még észre, hogy az egyetlen testre jellemző mennyiség, amit meg kell adnunk, az a csillag tömege. A **STELLAR** modul nyomon követi a fundamentális csillagparaméterek fejlődését 1,4 naptömeg alatti gázóriásoknál. Ide tartozik az effektív hőmérséklet, sugár, luminozitás, XUV luminozitás, valamint forgási ráta. A modul a már említett, Baraffe és munkatársai által létrehozott, Napunkéhoz hasonló fémességű csillagokra alkotott modellekre végez köbös interpolációt a csillagtömeg és idő figyelembe vételével, így modellezve a paraméterek fejlődését [9]. Ilyen értelemben a futtatásokból kinyerhető adatok egy-egy függvény adott időpillanatban vett értékei.

Ezek után vessünk egy pillantást a főfájlra is:

```
sSystemName             system
iVerbose                5
bOverwrite              1
saBodyFiles             star.in
sUnitMass               solar
sUnitLength             AU
sUnitTime               YEARS
sUnitAngle              d
bDoLog                  1
iDigits                 6
dMinValue               1e-10
bDoForward              1
bVarDt                  1
dEta                    0.01
dStopTime               10e9
dOutputTime             1e9
```

Az általam felhasználható adatok szempontjából az utolsó kettő kapcsoló tartozik a legfontosabbak közé. A másik fájlban megadhattuk a csillag kezdeti korát, ebben pedig azt a végső állapotot, amit szeretnénk, hogy elérjen a szimuláció végén. Ezért a *dStopTime* paraméter felelős. Rögtön utána láthatjuk a *dOutputTime* értéket, ami a kimeneti fájlban látható lépésközt jelöli. Nem összekeverendő a *dEta* változóval, ami a szimuláció mintavételezési sűrűségét, vagyis felbontását, pontosságát szabályozza. Jelen felállásban a csillag 1 millió évről ugrik rögtön 1 milliárd évre és egyenletes lépésközzel érkezik majd el 10 milliárd évhez.
Ezzel az elképzeléssel elkészítettem egy olyan adatsort, amely cél szempontjából identikus a már meglévő 3. táblázat adataival. Fontos megjegyezni, hogy a tesztek a fent látható beállításokkal futottak le, a következő kivétellel: a csillag korát, a kimeneti felbontást, a megállási időt variáltam, emellett pedig kikapcsoltam a fősorozat elhagyásakor történő leállásért felelős paramétert. A kapott eredmény a(z) 2. táblázatban és a(z) 4.4. ábrán látható.

Kor[Gyr]	Sugár[$R_\odot$]	T_{eff}[K]
1	0.943	5626
2	0.966	5661
3	0.993	5696
4	1.026	5730
5	1.069	5763
6	1.122	5782
7	1.186	5772
8	1.213	5767
9	1.213	5767
10	1.213	5767

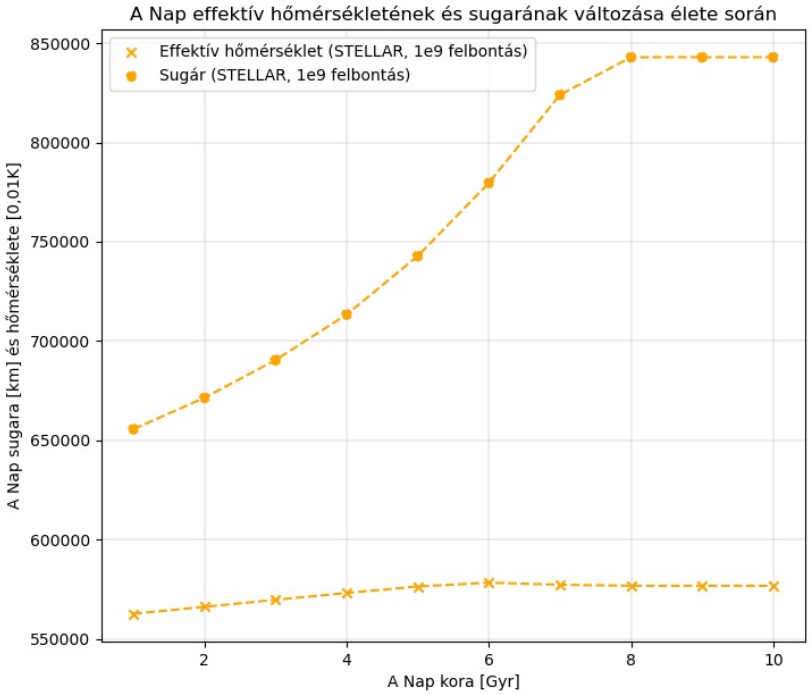
2. táblázat. A becsült sugár- és hőmérsékletértékek Napunk fejlődése során.

Hogy jobban össze tudjuk hasonlítani az egyes eredményeket, tekintsük meg a(z) 4.5. ábrát.

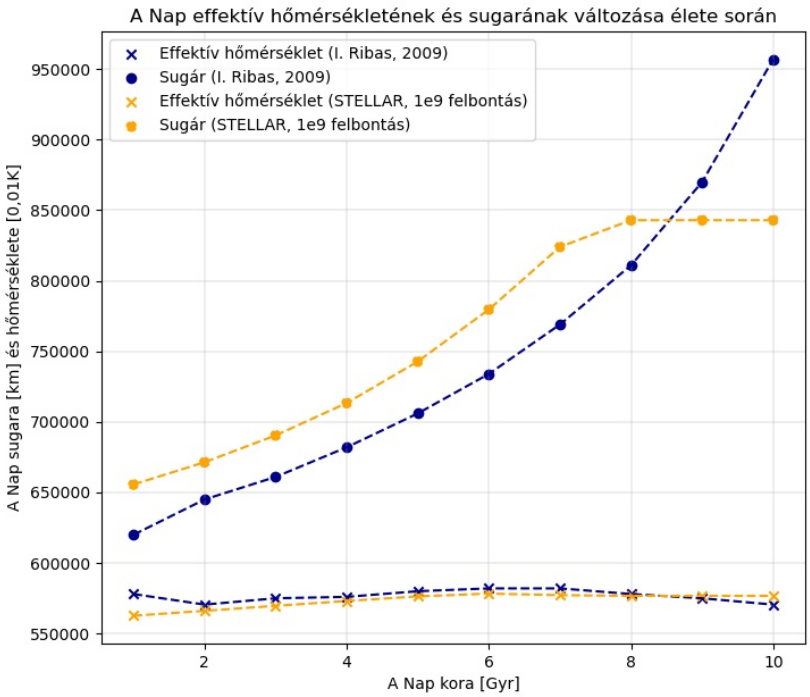
Észrevehetjük, hogy 8 milliárd évtől jelentősen eltér a sugarat mutató görbék íve, ez pedig a **VPlanet**-tel nyert információból fakad. Ha megnézzük a(z) 2. táblázatot, láthatjuk, hogy az utolsó néhány értékpár megegyezik. Hogy a hiba mibenlétére fényt derítsek (valamint hogy további használható információt nyerjek ki saját programomhoz), többször elvégeztem a futtatásokat, különböző bemeneti értékekre.

Az alább látható 4.6. ábrán a kiegészített grafikon, rajta az előzővel megegyezően lefedett intervallum, egy nagyságrenddel nagyobb felbontásban (vagyis 100 millió éves lépésközzel).

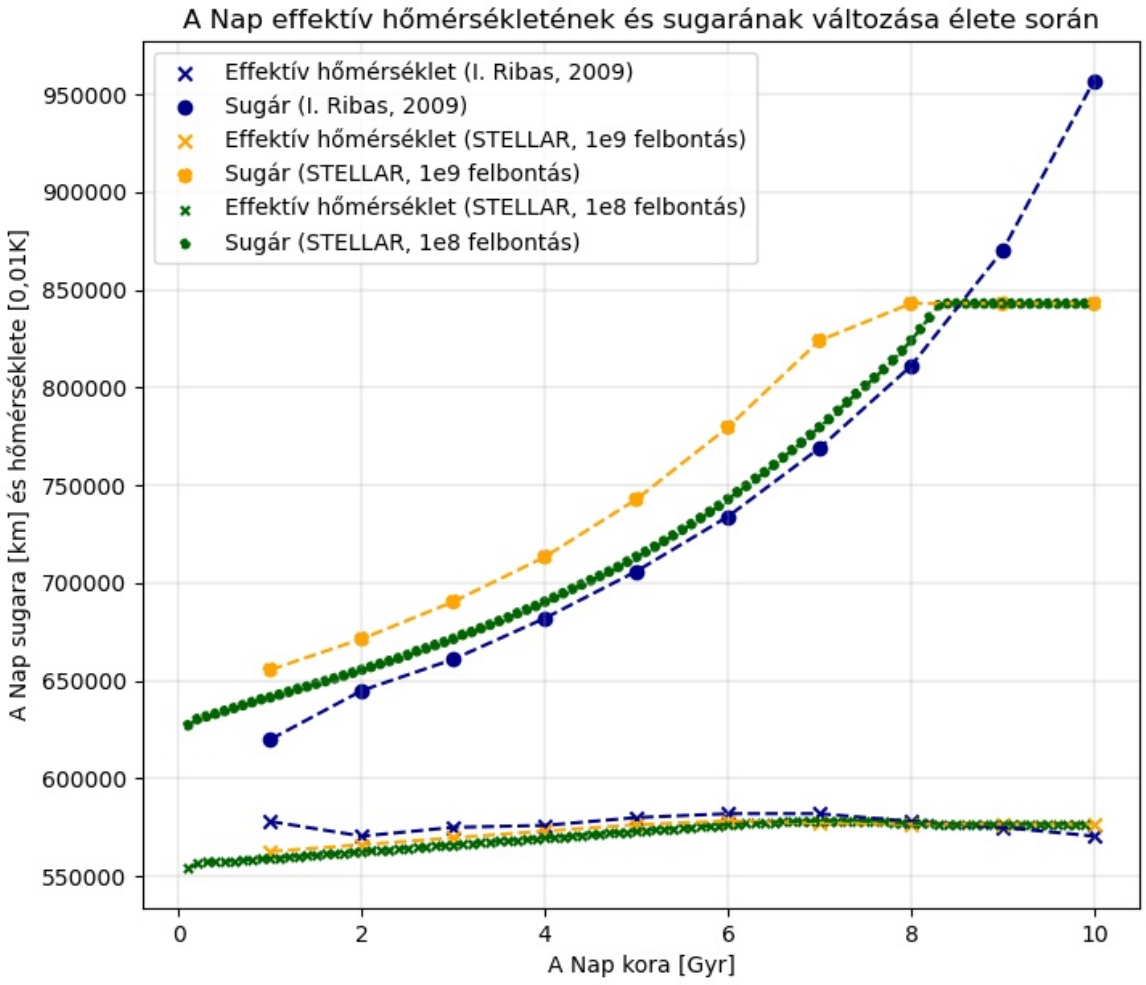
Látszólag a hiba szisztematikus, nagyjából 8,2 milliárd évnél történik az értékek levágása/bekonstansodása. A készítők cikkében és dokumentációjában nem találtam erre vonatkozó információt, ezen probléma körbejárása további, esetleges posztgraduális munkám részét képezheti.



4.4. ábra. A **STELLAR** modul segítségével előállított adatsorral készített grafikon.



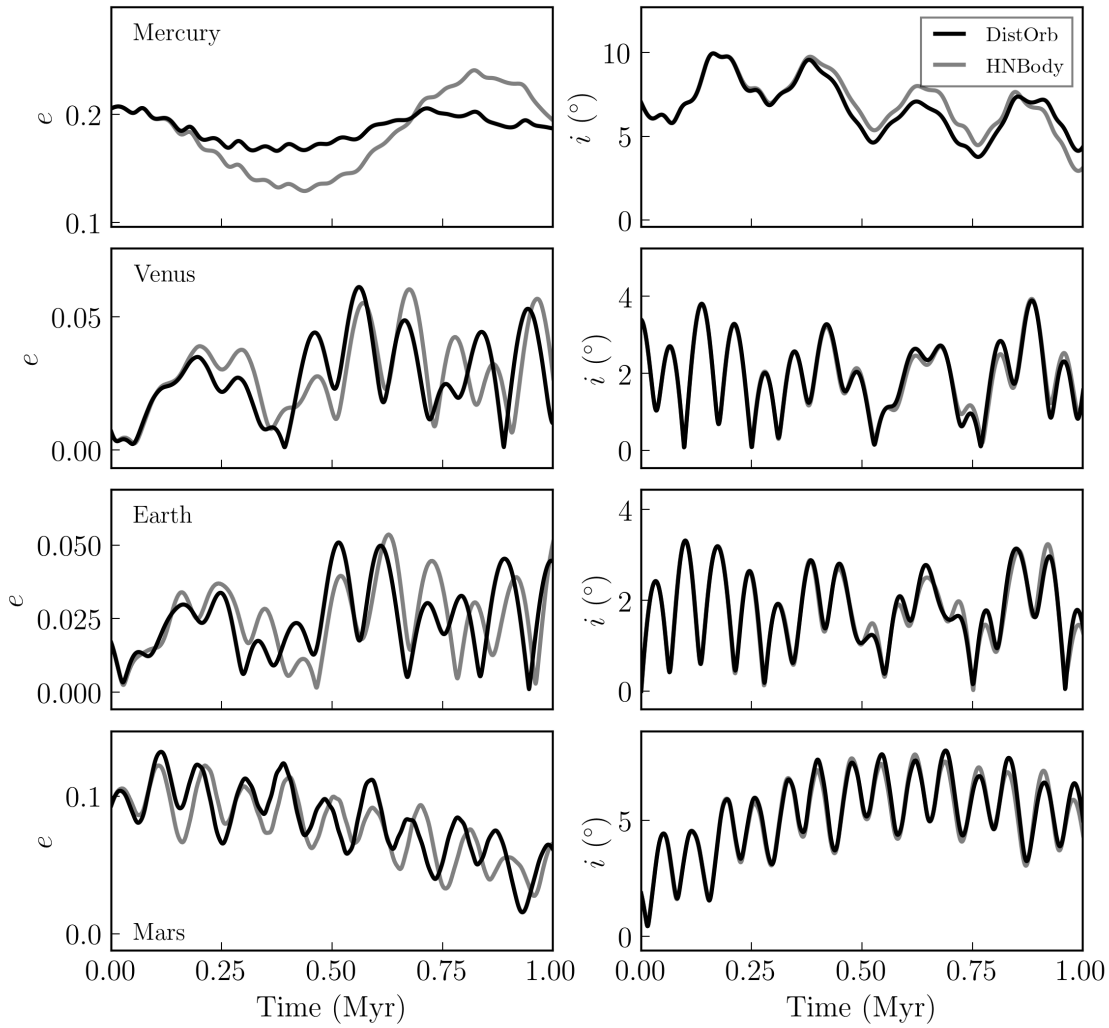
4.5. ábra. A **STELLAR** modul segítségével, valamint a már látott, I. Ribas (2009) cikke alapján kinyert értékek összehasonlítása.



4.6. ábra. A **STELLAR** modul segítségével, valamint a már látott, I. Ribas (2009) cikke alapján kinyert értékek további összehasonlítása.

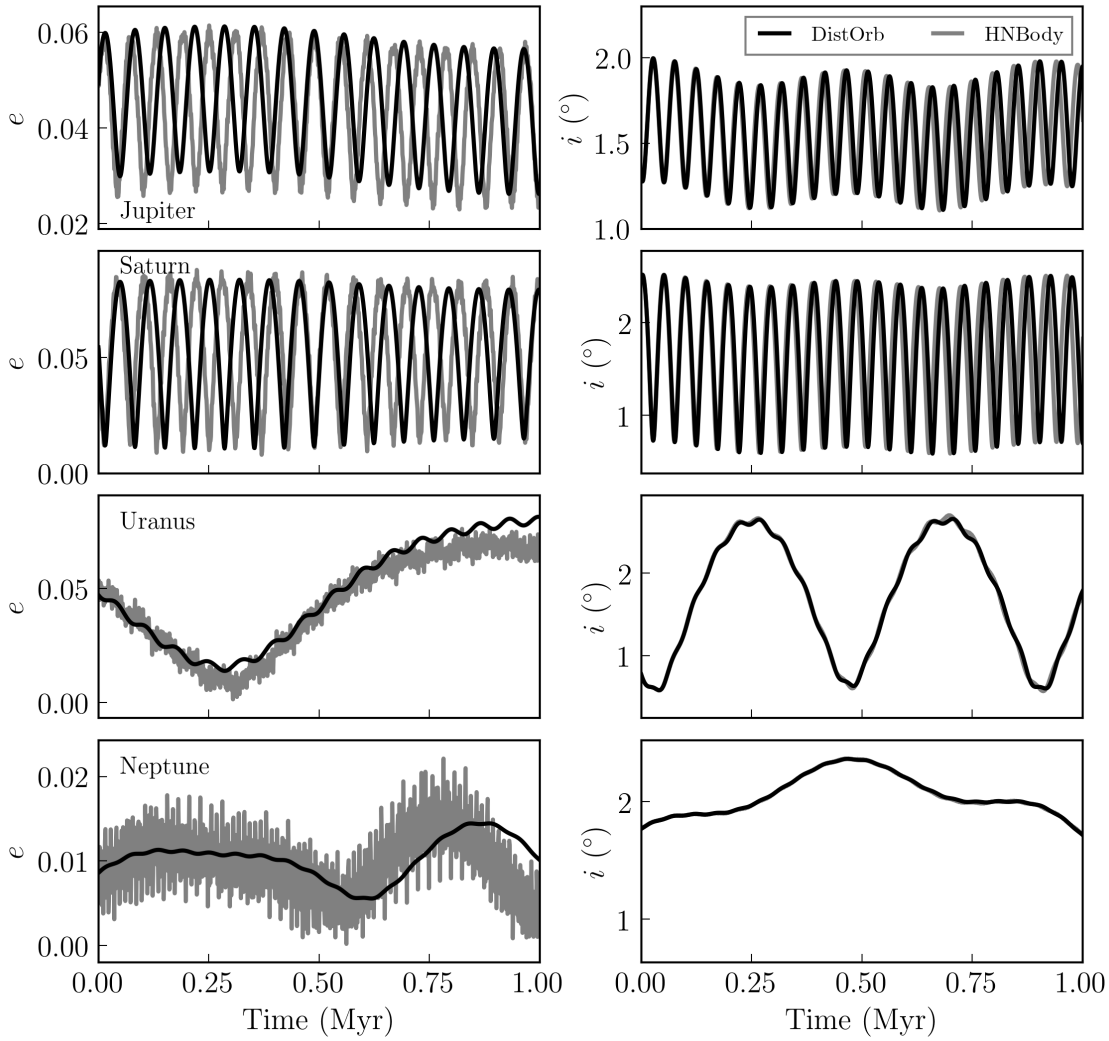
4.5.3. "Solar System Orbital Dynamics from Secular Theory"

Az adott, fejezetcímben is olvasható probléma eredményére alkotott grafikonok igencsak ígéretesnek tűnhetnek, ha csillagrendszer-fejlődést kutatunk. Ahogyan az a(z) 1. táblázatban is látható, nem csak egy futtatást eszközöltem. Elsősorban az alapbeállításokat megtartva sikeresen elvégeztem a tesztet, gond nélkül visszkapva a várt eredményt, ami a(z) 4.7. és 4.8. ábrákon megtekinthető.



4.7. ábra. A **DISTORB** és **DISTROT** modulok segítségével szimulált probléma eredményének ábrázolása. Bal oldalt az egyes belső bolygópályák excentricitásának, jobb oldalt pedig inklinációjuknak változását láthatjuk az idő függvényében.

Fontos megjegyezni, hogy a próbaszimuláció időintervalluma mindössze egymillió évet ölel fel, így a nagyléptékű pályaparaméter-változásokról nem kapunk képet, illetve az adott problémába nem vették bele a Nap fejlődését, ami úgy nyilvánult meg egyszerűen, hogy kihagyták a **STELLAR** modult. Ezáltal felbátorodva igyekeztem a saját szimulációm megalkotni, melyben figyelembe vesszük a csillag evolúcióját és nagyobb léptékekben haladunk, s vizsgáljuk, miképpen módosulhatnak a bolygópálya-excentricitások és -inklinációk. Több próbát is tettem, egészen 1 milliárd évtől 10 milliárd éves tartományon is, viszont ahogyan azt vártam, a program túl sokáig futott, így sajnos azt le kellett állítanom és elvetni az ötletet. Az eredeti szimuláció fájljaiba belepillantva tudomást szerezhetünk a mintavételezési sűrűségről és egyéb technikai jellegű információkról:



4.8. ábra. A **DISTORB** és **DISTROT** modulok segítségével szimulált probléma eredményének ábrázolása. Bal oldalt az egyes külső bolygópályák excentricitásának, jobb oldalt pedig inklinációjuknak változását láthatjuk az idő függvényében.

```
bDoForward      1
bVarDt          1
dEta            0.01
dStopTime       1000000
dOutputTime     1000
```

Az utolsó három paraméter rendre továbbra is a számítások sűrűségét, a megállási időt, valamint a kimeneti időközt jelenti. Jelen felállásban a szimuláció néhány percet vesz igénybe, viszont a *dStopTime* kapcsolót nagyságrendekkel magasabb értékre állítva a futási idő is jelentősen megnövekszik. Ezt valamivel kompenzálni lehet a mintavételezés sűrűségének csökkentésével, viszont többszöri próbálkozásra sem sikerült olyan konfigurációt létrehozni, amely egy óránál kevesebb időt venne igénybe, az annál hosszabbakat - tehát az összes futtatást - pedig leállítottam. Konklúzióként az vonható le, hogy egy ilyen probléma szimulálására érdemes egy középrendű asztali számítógépnél erősebb és gyorsabb felszereléssel rendelkezni.

5. Saját kódjaim fejlesztése

Ez a fejezet két részt tartalmaz, melyből az első a BSc-szakdolgozatomban elért eredmények esztétikai és tartalmi továbbfejlesztéséről szól. A másik alfejezetben egy teljesen új megközelítést mutatok be, ugyanis a diplomamunkám második felében létrehoztam egy grafikus kezelőfelületet, így nem csak terminálból lehet már használni a programot. További előny, hogy a három kód már nem külön-külön, hanem egy egyesített szkript futtatásával elérhető.

5.1. Eredeti környezet

Míg igyekeztem ötleteket meríteni a **VPlanet** készítői által publikált cikkből, számos problémába botlottam. Az egyes folyamatok bonyolultsága meghaladja azt a szintet, ahol még a két projektet össze lehet fűzni. Emiatt megpróbáltam először kisebb lépéseket tenni a fejlődés felé. Az 1. fejezetben bemutatott programok igen sok revizionálást és újradolgozást igényeltek mindidáig. A három közül a tízpaneles, evolúciót szemléltető kód (mostantól EHZ - *Evolution of the Habitable Zone*) fejlesztése a grafikus környezet létrehozásáig szünetelt, arra a későbbiekben tértem rá. Addig viszont fontosabbnak tartottam a másik két kód újradolgozását. Ezek továbbra is: egy egyszeri időállapotot ábrázoló (mostantól HZ - *Habitable Zone*), valamint egy időintervallum alatti változást vizsgáló program (mostantól CHZ - *Continous Habitable Zone*). Hogy az újításokról szó eshessen, az eddigi funkciókat mutatom be előbb röviden.

Mint azt már említettem, ahogyan az EHZ, úgy a CHZ is egy többsoros adatfájlt olvas be, aminek csillagparamétereket kell tartalmaznia. Oszlopok szerint:

- csillagrendszer adott állapothoz tartozó kor (számításoknál nincs jelentősége, csak az ábrán történő címkézésnél, enélkül is fut a program);
- központi csillag sugara (megadható km-ben és napsugárban is egyaránt);
- központi csillag effektív hőmérséklete (megadható K-ben)

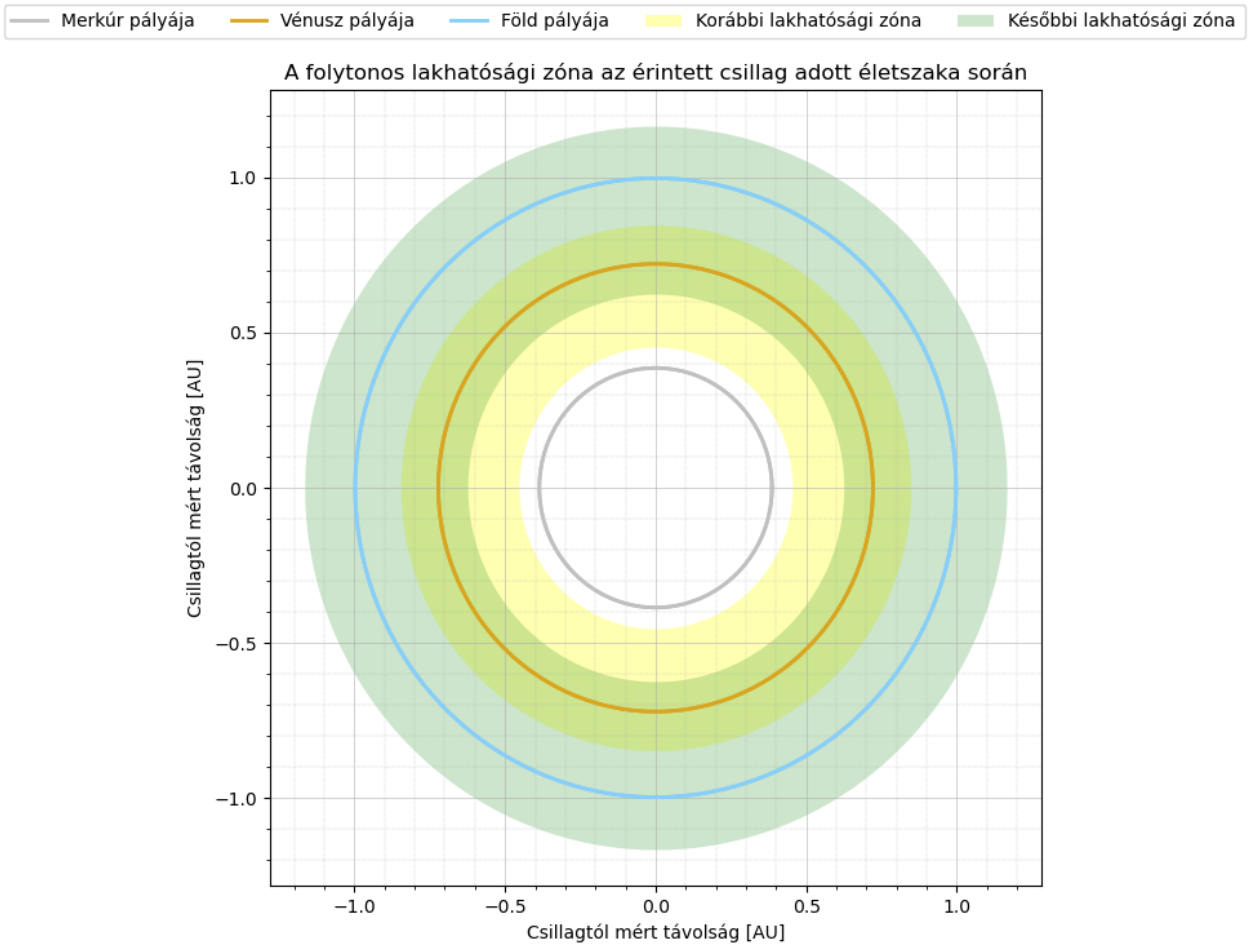
A CHZ program indításkor több kérdést is feltesz a felhasználónak, ezzel biztosítva és ellenőrizve a helyes működést. A kérdések a következők:

- 1 - "Gyr-ben vagy Myr-ben van a kor?"
- 2 - "km-ben vagy napsugárban van a csillag sugara?"
- 3 - "mekkora albedóval számoljon a program?"
- 4 - "melyik fájl legyen beolvasva (ezután az adatfájl tartalma kiírásra kerül)?"
- 5 - "melyik legyen a kezdeti állapot?"
- 6 - "melyik legyen a végső állapot?"

Természetesen nem megfelelő inputok esetén a program figyelmezteti a felhasználót a beviteli hibára. Számos újításra volt szükség, hogy funkciókban gördülékenyebb, kinézetre átláthatóbb és letisztultabb legyen, valamint valósághűbb képet adjon nekünk ez a szoftver.

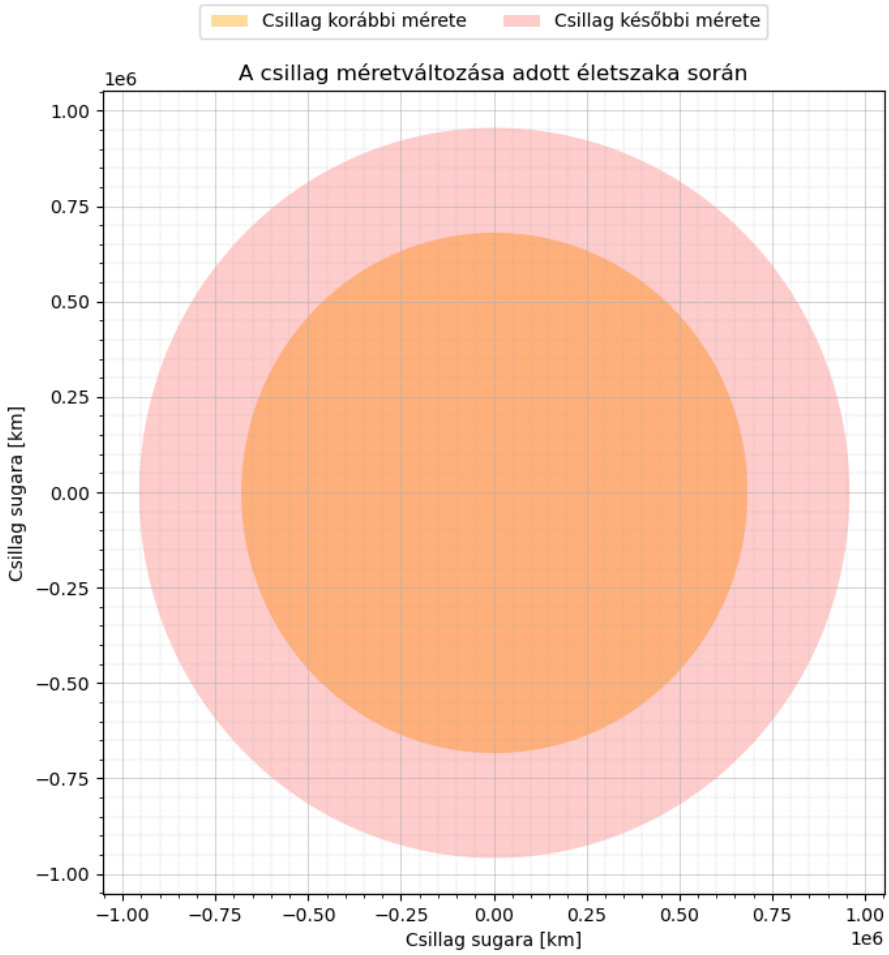
Első ötletként egyesítettem a HZ-t a CHZ-vel. A kód némi módosításával, a két program egyként tud funkcionálni. A hatodik kérdésre "-" válasszal csak egy zóna lesz ábrázolva, így a program HZ módba kapcsol vissza. Ha a válaszunk mégis egy másik szám, ugyanúgy CHZ-ként funkcionál tovább a kód.

Ezután fontosnak tartottam, hogy egy újabb külsőt kapjanak a zónák, így a felhasználó könnyebben átláthatja a paneleket. A letisztultabb kép folytonos zónára a(z) 5.9. ábrán látható, míg ugyanez egy egyszerű lakhatósági zónára a(z) 8.2. ábrán tekinthető meg.



5.9. ábra. Az új külsőt kapott kimenet CHZ esetre nézve.

Mivel a csillag paramétereit mindenképp meg kell adnunk, így érdekes lehet feltüntetni a gázóriást az ábrán, összehasonlításképp a zónához. Ámbár sajnos nem összemérhető a kettő, a csillag méretének tízszerese is csak nehezen látszik napszerű esetben. Emiatt úgy döntöttem, hogy kibővítem az ábrát egy újabb panellel. Bal oldalt továbbra is az eddigi eredmények láthatóak, míg jobb oldalt megjelenik a csillag mérete (a két panel külön skálával rendelkezik, nem méretarányosak), ez pedig akkor lehet releváns, ha időbeli fejlődést vizsgálunk. Erre egy jó példát láthatunk a(z) 5.10. ábrán.



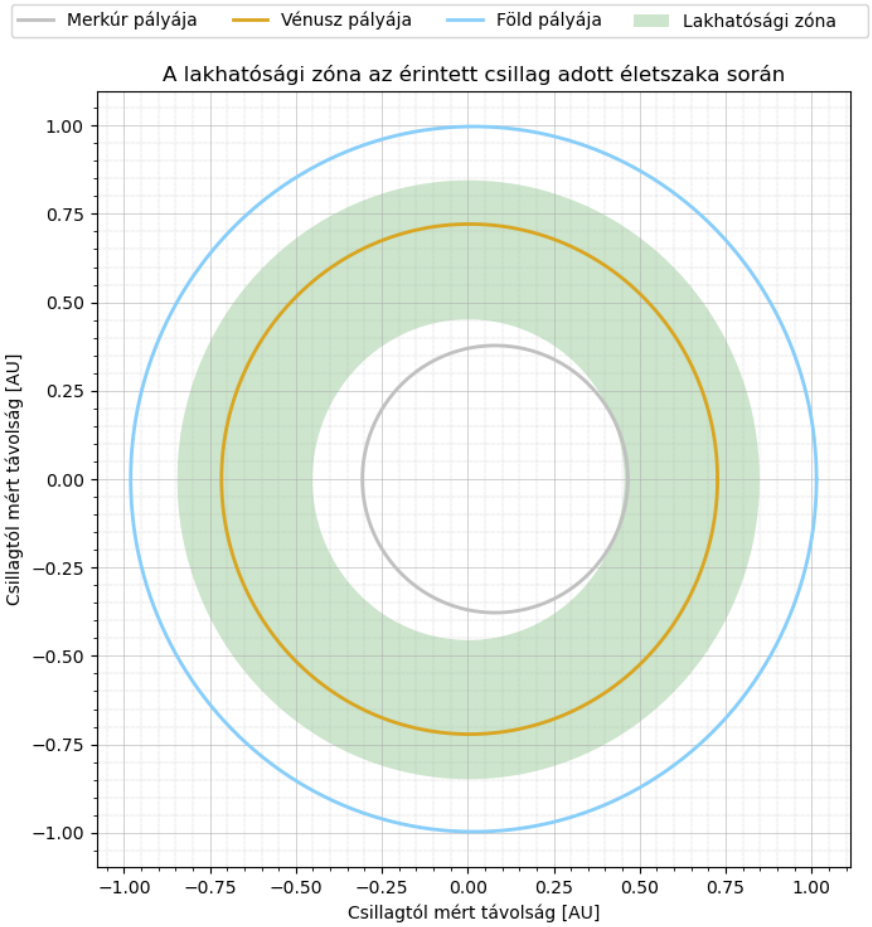
5.10. ábra. Az új panel, amelyen a csillag méretváltozását figyelhetjük meg.

Ezt követően opcionálissá tettem a Naprendszerünkhöz tartozó bolygók ábrázolását, mivel ez néha problémát okozhat. Például ha egy TRAPPIST-1-hez hasonló rendszert vizsgálunk és szeretnénk referenciát, akkor a program mindenképp megmutatja nekünk a Merkúr pályájának nagyságát, emellett viszont eltörpül a rendszer, amit vizsgálni szeretnénk. Így egy újabb kérdéssel bővült a CHZ:

- 7 - "legyen referencia?"

A jövőben akár tervben van még az adott funkció elhagyása, ehelyett még egy panellel bővíteném az ábrát: a rendszer referenciával és anélkül.

Ezután a tudományos bővítés terén első lépés volt a körpályák elhagyása. Kisebb-nagyobb módosítások véghezvitelével a körfüggvényeket ellipszisfüggvények váltották fel, a bolygó osztály pedig új változóval gyarapodott - ez az excentricitás. Naprendszerbeli bolygók esetén a Merkúr és a Mars viszonylag excentrikus égitestek, így ilyen esetekre remekül látszik, hogy a program helyesen interpretálja az újítást. Ennek eredményét a(z) 5.11. ábrán láthatjuk.



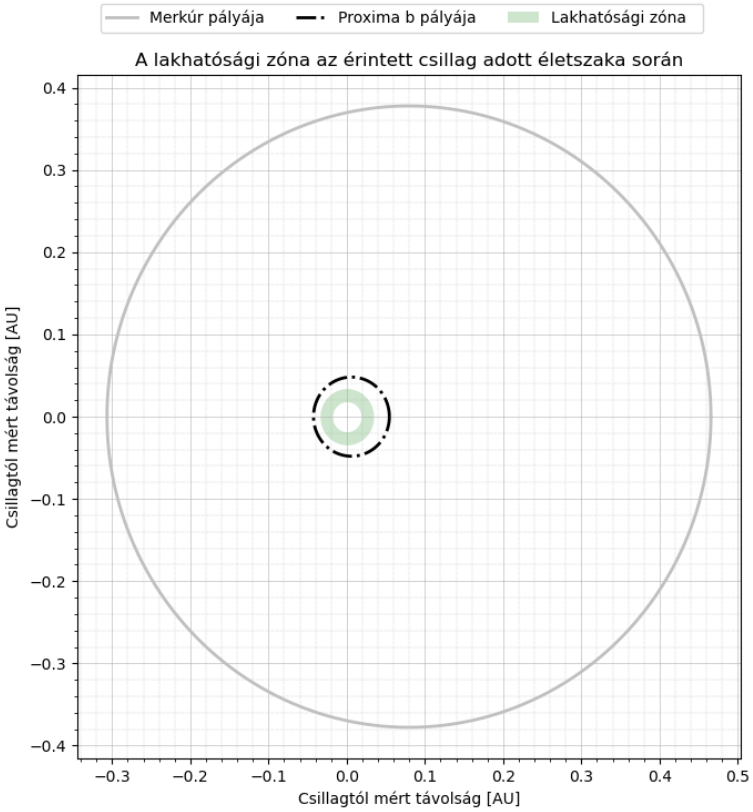
5.11. ábra. Az excentritásokkal ellátott bolygópályákat megjelenítő kód - jelenleg egy időpillanatra.

A diplomamunka alatt bevezetett végső funkció a terminálos futtatáshoz pedig a tetszőleges planéta ábrázolása. Ehhez három új kérdést tesz fel a program:

- 8 - "a bolygó neve?"
- 9 - "a bolygó félnagytengelye(AU)?"
- 10 - "a bolygó excentricitása?"

Ha nem kívánunk plusz bolygót feltüntetni, egyszerűen csak üresen kell hagyni a választ. Ezzel a funkcióval lényegében különféle szcenáriók érhetőek el, melyekre egy példa a következő:

Kikeresünk megbízható forrásból egy-egy csillagadatot a bementi fájlhoz, majd ugyanezt megtesszük az adott bolygójára, ezek mellé pedig tetszőlegesen kérhetünk Naprendszerből vett referenciát. Így közelítésekkel használt, többnyire valós képet kaphatunk. Tekintsük most a szomszéd rendszert, a Proxima Centurit [10][11]. Mivel a pályák jóval kisebbek a saját Naprendszerünk méreteinél, így a program eleve a legbelső bolygót tünteti fel, kért referencia esetén [12]. A probléma ábrázolva a(z) 5.12. és 5.13. ábrákon látható.

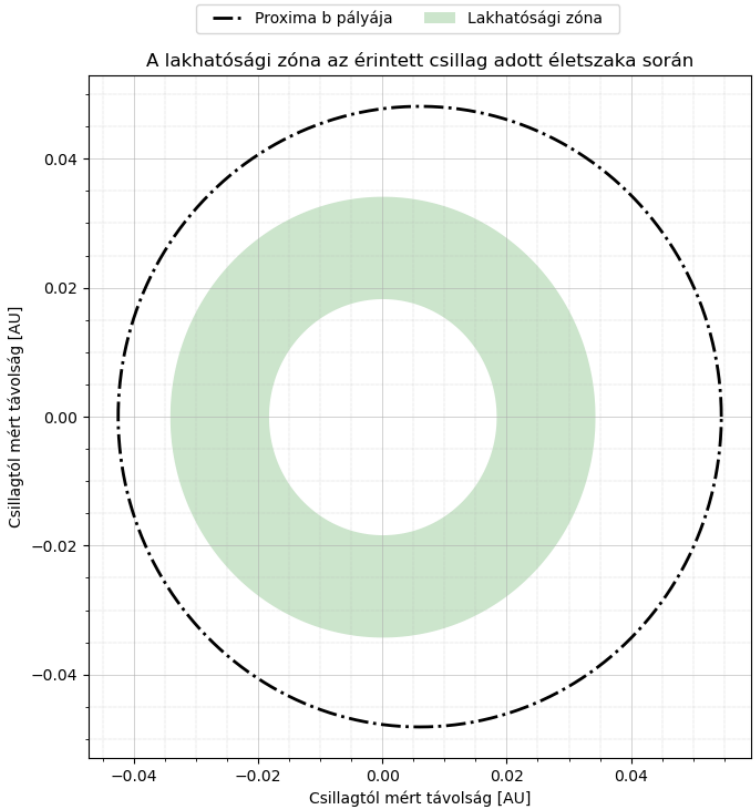


5.12. ábra. A szomszéd rendszerünk (Proxima Centauri) ábrázolva referenciával. Lévén a gázóriás M típusú flercsillag, jóval kisebb mértékű sugárzás jellemző rá, így a körülötte lévő lakhatósági zóna igen közel van hozzá, továbbá jelentősen vékonyabb Napunkéhoz viszonyítva. Ehhez képest lett feltüntetve a Merkúr pályája, szimbolizálva a méretbeli különbségeket.

A jövőben ezt a funkciót igyekszem majd kibővíteni olyan módon, hogy akár adatfájlból is be lehessen olvasni egy egész rendszert, ne pedig csak egy csillagot és annak egy bolygóját. A következő részben viszont láthatjuk, hogy ez a funkció nem került bele a továbbfejlesztett verzióba, egyrészt időhiány, másrészt pedig a kód kompaktsága miatt, ámbár a későbbi munkám folyamán tervezem az ilyen irányú bővítést is.

5.2. Grafikus kezelőfelület

Különböző ötletektől felbátorodva sokat tanulmányoztam a Python nyelv objektumorientált lehetőségeit, hiszen ez számomra a jövőben is nagyon hasznos lesz, így igyekeztem egy ilyen irányba elvinni ezt a projektet is. Az említett nyelvnek több nagyobb könyvtára van, melyekkel grafikus felületeket (mostantól GUI, vagyis *graphical user interface*) lehet felállítani és személyre szabni. Vannak beépített modulok, amikkel viszonylag rövid idő alatt akár professzionálisnak tűnő felületeket lehet létrehozni (egyik legismertebb a tkinter), viszont az átható megértéshez és tudáshoz érdemes ötletnek tartottam, hogy magam próbáljam meg a saját fejlesztésű GUI-m megalkotását. A feladatra különféle fórumokon olvasott információk alapján a *pygame* könyvtárat választottam, ugyanis saját megítélés és számos pozitív vélemény alapján funkciókban ez bővelkedik a leginkább. Emellett nagyon fontos a további gyakorlat szerzése az osztályokról és azok funkcióiról. A kódot több részletre is fel lehetne bontani, viszont nagyjából egységes függvényekben és osztályokban, így az egész elfér egyetlen fájlban.



5.13. ábra. A szomszéd rendszerünk (Proxima Centauri) ábrázolva referencia nélkül. Ha közelebbről szeretnénk megvizsgálni a csillagrendszert, érdemes ilyen esetben a referenciát kikapcsolni, így "ránagyítva" az adott zónára. A fent részletezett okokból kifolyólag jelenleg csak egy extra bolygót tudunk hozzáadni a kérdéses problémához.

5.3. A GUI felállítása

Az eredeti terv egy része, amit meghatározó irányként követtem, az a szakdolgozat alatt készített programjaim egyesítése, valamint felhasználóbarátabbá tétele. Két fontos modul, amit meg kell hívni, a már említett *pygame* és a *time*, utóbbiról később látjuk majd, miért. Ezután inicializálni kell a meghívott könyvtárat és felállítani a kérdéses ablakot. Mivel módonként eltérő számú bemeneti lehetőségek vannak, valamint a kimenet is jelentősen eltér ezen esetekben, így több képernyőméretet is definiáltam. A legelső, ami fogad minket, az alábbi módon áll fel:

```
import pygame, time
...

pygame.init()

W0 = 700
H0 = 600

display = pygame.Surface((W0,H0))
screen = pygame.display.set_mode((W0,H0))

pygame.display.set_icon(pygame.image.load("planet.png"))
```

```
pygame.display.set_caption("Habitable zone calculators")

clock = pygame.time.Clock()

black = (0,0,0)
white = (255,255,255)
loop1 = True
```

Kevésbé fontos, mégis megemlítendő részlet, hogy az adott alkalmazást szabadon elnevezhetjük és elláthatjuk saját ikonnal is, mint ahogyan az lentebb látható. Ha ilyen módon szeretnénk futtatni a kódot, természetesen egy irreszponzív, üresen megjelenő ablakot kapnánk, ugyanis nem adtunk meg billentyűleütéssel és/vagy egérkattintással működő bemenetet, valamint semmi érdemlegeset nem rajzoltunk a képernyőre. Hogy egy teljesen működőképes programunk legyen, ciklusba kell foglalnunk azt, amit valós időben látni szeretnénk a képernyőn. A főmenü számára alkotott ciklus a kódban a *loop1* nevet viseli. A képernyő az alábbi adalékkal már gond nélkül lefut, viszont még továbbra is üres marad:

```
while loop1:
    screen.fill((200,200,200))
    clock.tick(60)

    display.blit(screen, (0, 0))
    pygame.display.update()
```

Mielőtt bármit kiíratnánk a képernyőre, a lehető legeffektívebb módon meg kell alkotni egyes osztályokat, amiknek az objektumaival interaktálni tudunk programfuttatás közben. A régebbi munkámban is megtalálható már egy bizonyos osztály, amely a *planet* névvel lett ellátva, viszont akkor nem részleteztem a működésének mibenlétét a biztosabb tudás hiányában. Ennek az elmaradását a következő sorokban pótlom:

```
class planet():
    def __init__(self, name, d, color, ecc):
        self.name = name
        self.d = d
        self.color = color
        self.ecc = ecc

mercury = planet("Merkúr", 57910000, "silver",0.206)
venus = planet("Vénusz", 108200000, "goldenrod",0.007)
earth = planet("Föld", 149600000, "lightskyblue",0.017)
mars = planet("Mars", 227900000, "salmon",0.093)
jupiter = planet("Jupiter", 778500000, "sandybrown",0.048)
saturn = planet("Szaturnusz", 1433400000, "khaki",0.054)
uranus = planet("Uránusz", 2870970000, "paleturquoise",0.047)
neptune = planet("Neptunusz", 4498200000, "dodgerblue",0.009)
```

Amit fentebb láthatunk, az a következő. Létrehoztam a *planet* osztályt és inicializáltam azt. Ezután az osztályba tartozó, leendő elemeket elláttam tulajdonságokkal. Minden, az adott osztály részét képező elem rendelkezik névvel, félnagytengellyel, színnel, valamint excentricitással. Pár sorral lejjebb már rögtön láthatjuk, hogyan hozok létre különböző bolygókat a kézzel beírt paraméterek segítségével. Ezekre később hivatkoztam és hivatkozom a kódban,

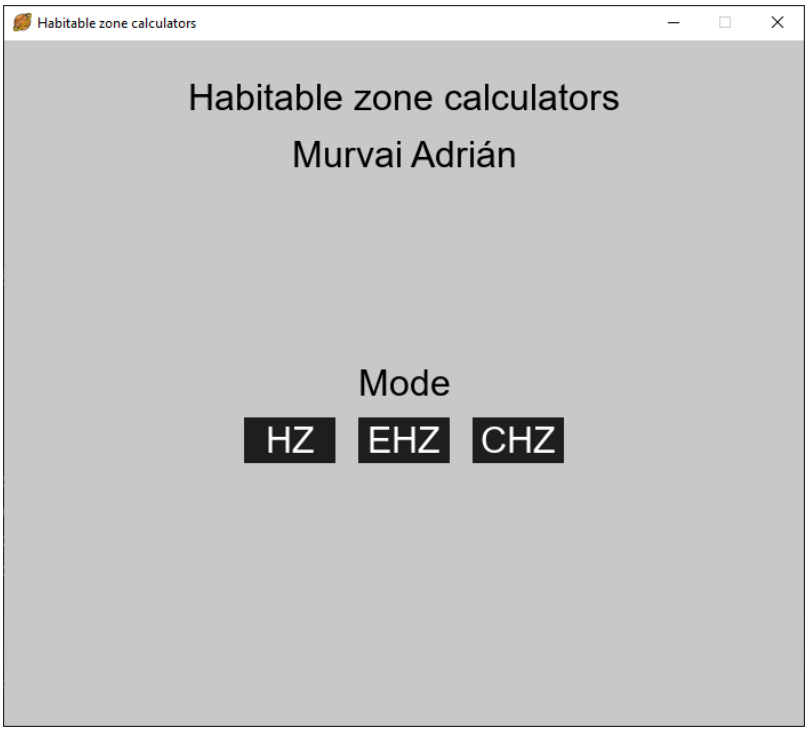
amikor saját rendszer alapú referenciát kérek a futtatáshoz. Természetesen ez az osztályok használatának a legegyszerűbb módja. A következő kettő több szerepet kapott, hiszen a GUI szerves részét képezik. Ezek a *Text* és *Button* osztályok. Az előbbi a következőképpen néz ki:

```
class Text:
    def __init__(self, text, x, y, w, h, size):
        self.text = text
        self.w, self.h = w, h
        self.x, self.y = x, y
        self.surf = pygame.Surface((w, h))
        self.surf.fill((200, 200, 200))
        basic_font = pygame.font.Font('arial.ttf', size)
        text_surface = basic_font.render(self.text, True, black)
        text_rect = text_surface.get_rect(center=(round(w / 2), round(h / 2)))
        self.surf.blit(text_surface, text_rect)
        self.rect = self.surf.get_rect(center=(x, y))

    def maketext(self):
        screen.blit(self.surf, self.rect)
```

Ahhoz, hogy egy szöveg-objektumunk legyen, meg kell adni magát a szöveget, a pozíciójának koordinátáit, a szövegdoboz méretét, valamint a betűméretet. Ezt követően a szöveghez tartozni fog egy felület, ami a képernyőn fog megjelenni a kért helyen, amennyiben meghívjuk a hozzátartozó *maketext* függvényt. Ennél valamennyivel sokszínűbb a *Button* osztály:

```
class Button:
    def __init__(self, x, y, w, h, text):
        self.x, self.y = x, y
        self.w, self.h = w, h
        self.text = text
        self.bg_color = (30,30,30)
        self.text_color = (255, 255, 255)
        self.font = pygame.font.Font('arial.ttf', 32)

    def makebutton(self):
        surf = pygame.Surface((self.w, self.h))
        surf.fill(self.bg_color)
        text_surface = self.font.render(self.text, True, self.text_color)
        text_rect = text_surface.get_rect(center =
        (round(self.w/2),round(self.h/2)))
        surf.blit(text_surface,text_rect)
        self.rect = surf.get_rect(center =(self.x, self.y))
        screen.blit(surf, self.rect)

    def invert(self):
        self.text_color = (30, 30, 30)
        self.bg_color = (255, 255, 255)

    def reset(self):
        self.bg_color = (30,30,30)
        self.text_color = (255, 255, 255)
```

Viszonylag hasonlít a két osztály, ámbár utóbbinál, amennyiben meghívjuk az *invert* függvényt, a kattintható gomb háttérszíne, illetve szövegszíne az ellentettjére változik, így keltve a hatást, hogy aktiváltuk. A *reset* függvény ennek ellenkezőjét teszi, vagyis minden esetben az alapértelmezettre állítja a szükséges paramétereket. A fentebb részletezett ismeretek segítségével most már kibővíthetjük az előbbi kódrészletet annyira, hogy a szövegek és gombok megjelenjenek a képernyőn. Ehhez előbb definiáljuk a különböző elemeket a két osztályból, magában a ciklusban pedig meghívjuk őket a hozzájuk rendelt függvények segítségével. Az idetartozó kód, valamint a kapott ablak (5.14. ábra) alább látható.



5.14. ábra. A program elkészült főmenüje.

```
text_mode = Text("Mode", W0/2,300,80,40,32)
text_first = Text("Habitable zone calculators", W/2,50,600,40,32)
text_second = Text("Murvai Adrián", W/2,100,600,40,32)

bt_hz = Button(W0/2 - 100,350,80,40,"HZ")
bt_ehz = Button(W0/2,350,80,40,"EHZ")
bt_chz = Button(W0/2 + 100,350,80,40,"CHZ")
button_list = [bt_hz,bt_ehz,bt_chz]

mode = "HZ"

while loop1:
    screen.fill((200,200,200))

    text_mode.maketext()
    text_first.maketext()
    text_second.maketext()
```

```
    bt_hz.makebutton()
    bt_ehz.makebutton()
    bt_chz.makebutton()

    clock.tick(60)

    display.blit(screen, (0, 0))
    pygame.display.update()
```

Már csak a bemenetet kell megfelelően létrehozni és az ablak teljes mértékben reagálni fog az interakciókra. Az első ciklusban nincs szükség billentyűleütés általi jelre, így amit használunk, az kizárólag az egérgomb elengedésének a válasza. A következő részlettel bővül így a kód, a ciklus elején:

```
...
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            loop1 = False
            sys.exit()
        if event.type == pygame.MOUSEBUTTONUP:
            mouse_position = pygame.mouse.get_pos()
            for button in button_list:
                if (button.rect.collidepoint(mouse_position)):
                    button.invert()
                    mode = button.text
                    loop1 = False
...
```

A kód végighalad a *pygame* könyvtárban található összes lehetséges eseményen, ezekből pedig azokat az eseteket vizsgálja, amiket részleteztem feljebb. Ilyen például az ablak bezárása vagy a kattintás. Ezek után másik három ciklust vizsgál a program, mégpedig annak a függvényében, hogy melyik gombot aktiváltuk. Ezt láthatjuk az alábbi kódrészleten:

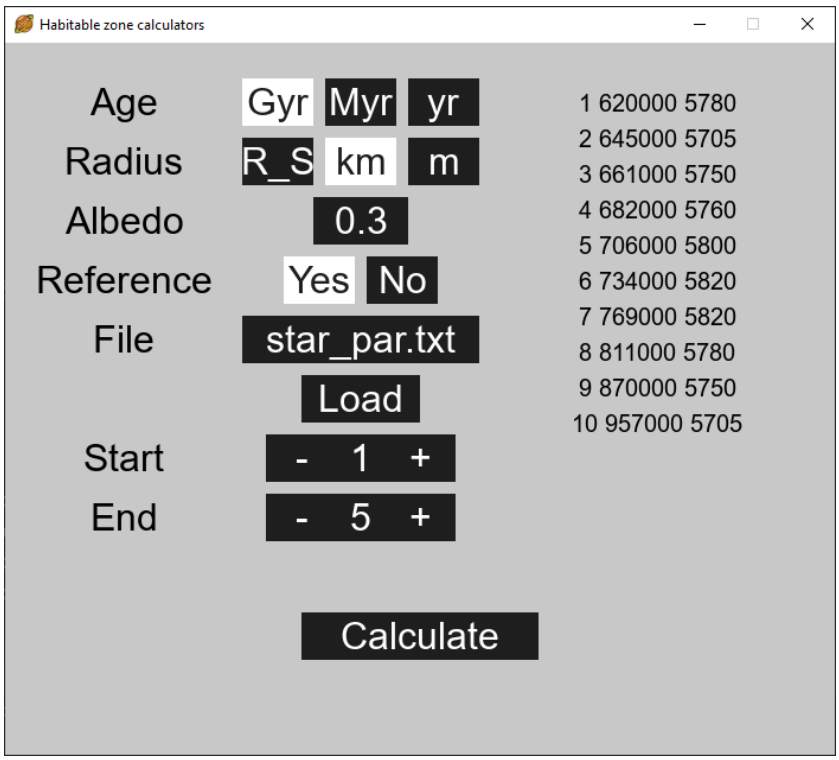
```
if mode == "HZ":
    loop2 = True
    loop3 = False
    loop4 = False
elif mode == "EHZ":
    loop2 = False
    loop3 = True
    loop4 = False
else:
    loop2 = False
    loop3 = False
    loop4 = True
```

A másik három ciklus szintaktikailag igencsak hasonlít az elsőre, így teljességükben azokat a csatolt linken tekinthetjük meg, viszont az eddig nem látott részleteket röviden ebben a fejezetben is szemléltetem [13]. Erre remek példa a mindhárom ciklusban fellelhető szövegdoboz, amibe mi magunk írhatunk egyrészt számadatot, másrészt pedig fájlnevet. Ezekre akkor van mód, ha a dobozba belekattintunk. Példaként a hőmérséklet beviteléért felelős részletet láthatjuk alább (albedóra és sugárra teljesen identikusak a feltételek).

```
if loop2:
    temp = "5000"
    ...
    bt_temp = Button(320,150,140,40,temp)
    ...
    numlist = [pygame.K_0, pygame.K_1, pygame.K_2, pygame.K_3,
               pygame.K_4, pygame.K_5, pygame.K_6, pygame.K_7,
               pygame.K_8, pygame.K_9, pygame.K_PERIOD]
    ...
    temp_writing = False
    ...
    while loop2:
        for event in pygame.event.get():
            ...
            if event.type == pygame.MOUSEBUTTONUP:
                mouse_position = pygame.mouse.get_pos()
                ...
                if bt_temp.rect.collidepoint(mouse_position):
                    temp_writing = True
                    bt_temp.invert()
                else:
                    temp_writing = False
                    bt_temp.reset()
                ...
            if event.type == pygame.KEYDOWN:
                ...
                elif temp_writing:
                    if event.key == pygame.K_BACKSPACE:
                        temp = temp[:-1]
                        bt_temp = Button(320,150,140,40,temp)
                    else:
                        if event.key in numlist:
                            if "." in temp:
                                if not event.key == pygame.K_PERIOD:
                                    temp += event.unicode
                            else:
                                temp += event.unicode
                            bt_temp = Button(320,150,140,40,temp)
```

A bevitel nagyon hasonló fájlnév esetére is, a számoknál lévő extra feltételt leszámítva. Ha minden paramétert sikeresen megadtunk, rákattinthatunk a megjelenő *Calculate* gombra, innenstől a kód nagyrészt ugyanúgy működik, ahogyan azelőtt, a kimenetet leszámítva. A CHZ módra jellemző menüt és annak bemeneti lehetőségeit megtekinthetjük a(z) 5.15. ábrán. Továbbá a HZ menü a(z) 8.3. ábrán látható.

A program beolvassa az indítás dátumát és idejét, valamint feljegyzi a kiválasztott módot, ezután elkészít egy ilyen nevű mappát, amibe elmenti az eredményeket. Rögtön ezután, amint a számítások véget értek, a kód belép a módhoz tartozó utolsó ciklusba, ami egy egyszerű képmegjelenítésért felel.

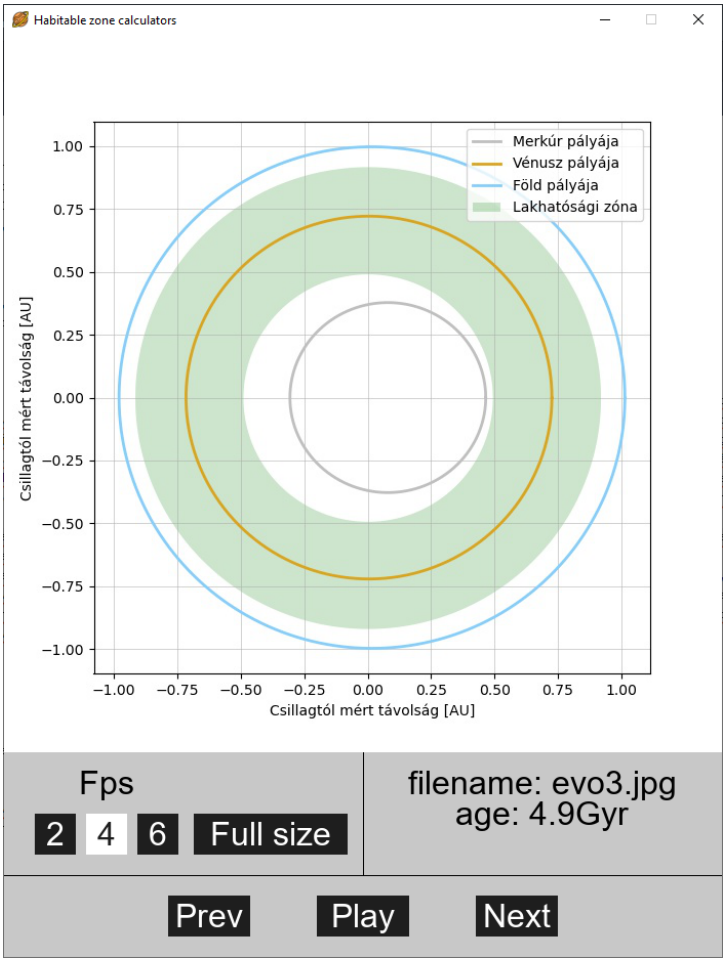


5.15. ábra. A CHZ menü beviteli lehetőségei. Az adott fájlt sikeresen beolvasva megjelenik a *Calculate* gomb, melyre rákattintva elindulnak a számítások.

```
date = datetime.now().strftime("%y%m%d_%H%M%S")
path = date
...
W4 = 700
H4 = 700
...
    plt.savefig(path+"/hz.jpg")
    run = True
    surf = pygame.image.load(path+"/hz.jpg")

    while run:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
                sys.exit()

        display = pygame.Surface((W4, H4))
        screen = pygame.display.set_mode((W4, H4))
        screen.fill((200,200,200))
        screen.blit(surf,(0,0))

        clock.tick(60)

        display.blit(screen, (0, 0))
        pygame.display.update()
```

5.4. A EHZ mód felújítása

Diplomamunkám készítéséig a fejezet címében is szereplő mód volt a legkevésbé csiszolt a három közül. Fontos feladatnak tartottam a hiányosságok bepótlását, így utólag a 5.1. fejezetben látható módosításokat itt is érvénybe helyeztem. Emellett újabb ötletektől felbátorodva egy teljesen új funkciót is kapott az lakhatósági zónák evolúcióját szemléltető mód. Kezdeti intuícióm volt, hogy két, egymástól független ablak futhasson, miután kiválasztottuk az EHZ-t, megadtuk az adatokat, majd a program elvégezte a számításokat. Sajnos ezt az ötletet el kellett vetnem, ugyanis a Python nyelv (jelenleg még) nem használható ilyesmire, egyetlen grafikus ablakra tudunk csak fókuszálni egy *.py* fájl futtatásával. A tervet kissé módosítva, az már megvalósíthatóvá vált. A kimenetnél nem csak egy, nagy, tízpaneles ábrát mentünk ki, hanem annak a tíz alképre felszabdalt változatát is, vagyis 10+1 fájlt. Ezután a GUI magát a teljes ábrát jeleníti meg, alatta egy *Animation* gombbal. Erre rákattintva teljesen új konfigurációt vesz fel az ablak, középpontban az első panellel. Lehetőségünk van az egyes időpontokban alkotott képek között előre és hátra haladni, valamint elindítani az animációt. Utóbbinak sebessége alapértelmezetten másodpercenként két képkocka, a sebességet manuálisan állítani is tudjuk, méghozzá 2, 4 és 6 értékek közt váltva. Kiíratásra kerül emellett még az adott fájl neve és a hozzá tartozó időpont is. A panelek és a teljes ábra között oda-vissza tudunk váltogatni, így ilyen formában egy teljesen megújult verzióját láthatjuk az EHZ módnak. A korábban említett *time* könyvtár itt is hasznosul, alább látható egy rövid részlet a lejátszásért felelős ciklusból. Emellett két képernyőmentést tekinthetünk még meg a szóban forgó módból. A teljes sorozat (elforgatva) a(z) 8.4., míg az animációs ablak a(z) 5.16. ábrán látható.

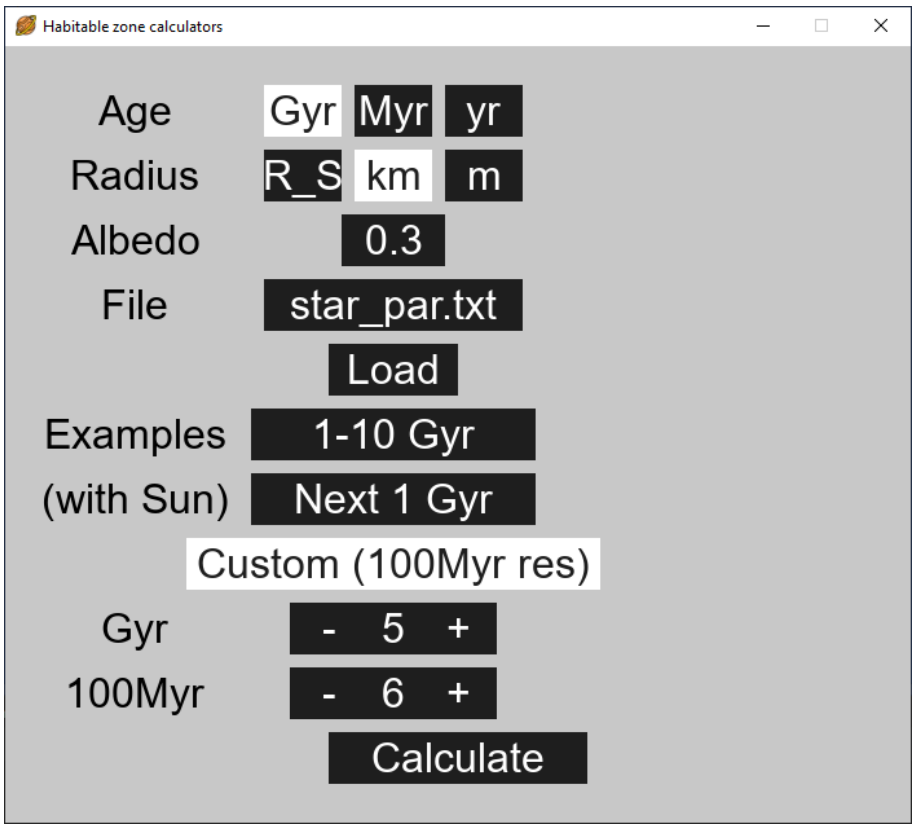


5.16. ábra. A felújított EHZ mód animációs ablaka.

```
playing = False
run = True
imglist = []

surf_full = pygame.image.load(path+"/evo_full.jpg")

for i in range(10):
    filename = path+"/evo"+str(i)+".jpg"
    imglist.append(filename)

i = 0
t0 = time.time()
fps = 2
...

while run:
    for event in pygame.event.get():
        ...
    ...
    clock.tick(60)
    realfps = 1000/fps

    if not playing:
        t0 = pygame.time.get_ticks()
    if playing:
        t1 = pygame.time.get_ticks()
        dt = t1 - t0
        if dt >= realfps:
            i += 1
            t0 = t1
            if i == 10:
                i = 0
    ...
```

5.5. A VPlanet-tel nyert eredmények hasznosítása

A(z) 2. táblázat adatai, valamint az itt - jelentős adatsorhosszra hivatkozva - nem közölt, jobb felbontású futtatás eredménye felhasználhatónak bizonyult a saját programomban is. Konkrét adatfájlt csak az EHZ és CHZ mód hasznosít, az új adatok beépítését pedig az előbbi menüjébe végeztem el. Míg eddig csak saját adatot tudtunk bevinni, addig ezentúl a szimulációkból kinyert végeredmények alapján előre definiált sémák vizsgálata is lehetséges. A(z) 5.17. ábrán láthatjuk az újabb funkció(ka)t.



5.17. ábra. A saját készítésű programom EHZ módjának - diplomamunka keretein belül értendő - végleges formája.

Mint azt fentebb láthatjuk, két, előre elkészített adatsorból létrehozott "preset"-et is választhatunk, ha az *Examples (with Sun)* paraméter melletti két gomb egyikére kattintunk, melyek a következők. A *1-10 Gyr* lehetőséggel ugyanazt az időintervallumot tudjuk lefedni, mint eddig, felhasználva a **VPlanet** eredményeit, mely a(z) 4.4. ábrán is látható. Emellett a következő egymilliárd évre számolt adatsorral is előállíthatunk egy ábrasorozatot, melynek felbontása 100 millió év. Ehhez a *Next 1 Gyr* gombra kell kattintanunk. Végül pedig a *Custom (100Myr res)* gombra kattintva saját magunknak is kiválaszthatunk egy 1 milliárd éves tartományt, ahol a 10 különböző, 100 millió éves időközű állapotot vizsgáljuk. Ennek alapja a már említett, nagyobb felbontású adatsor, melyről a készített grafikont a(z) 4.6. ábrán láthatjuk.

6. Összefoglalás és kitekintés

A diplomamunka mindkét vezérfonala igencsak hasznosnak bizonyult számomra. Dolgozatomban egy viszonylag új, talán nem túl sokak által ismert bolygó- és csillagrendszerfejlődéssel kapcsolatos szimulátorprogramot mutattam be, a **VPlanet**-et, vagyis a Virtaul Planet Simulator-t. Prezentálásra került a szükséges környezet inicializálása, a szoftver telepítése, valamint valamennyi, a felhasználó számára elkerülhetetlen fájl felépítése. Emellett saját magam bizonyosodtam meg, hogy az általam - rendszerszerűen - kiválasztott problémákra alkotott szimulációk és kódok megfelelően működnek, kisebb-nagyobb sikerekkel. A vizsgált problémákat feltüntettem, a sikertelen és problémás részeket pedig legjobb tudásomhoz hűen igyekeztem körbejárni, meghiúsulásuk okát megérteni. Igen fontos momentuma a végzett munkámnak, hogy az így előállítható eredmények közt volt olyan is, amit saját célra hasznosítani is tudtam (a **STELLAR** modul segítségével kinyerhető csillagfejlődési paraméterek), így sikeresnek könyvelhető el a **VPlanet** környezet általam történő megismerése, bemutatása, valamint felhasználása. A másik vezérszál, amiről beható leírást adtam a jelen dokumentumban, pedig a BSc-szakdolgozatom alatt megszerzett Python ismeretekkel elkészített ábrázolóprogramok újragondolása, továbbfejlesztése, illetve egyesítése. Míg az alapképzés keretein belül előzőleges tanulmányok nélkül dolgoztam és halmoztam fel elegendő tudást, úgy jelen munkámban az említett programozói ismereteket kétségkívül kibővítettem, az általam további irányként követendő szoftverfejlesztési útvonalat szem előtt tartva sikerként könyvelem el az eredményeket. A jövőben számos praktikus és hasznos program megalkotására leszek képes, akár tudományos, kutatással kapcsolatos, akár egyéb területeken.

Dolgozatom remek lehetőség volt az elsajátított ismeretek összegzésére és prezentálására, viszont mindig lehetne többet és többet bemutatni, ha a rendelkezésre álló idő nem lenne véges. Posztgraduális tervek közé tartozik a közérdekű célra megalkotott lakhatóságizónaábrázolóprogramom esetleges továbbfejlesztése, új funkciókkal kibővítve, lehetséges további érdekes, lakhatóságot befolyásoló faktorok figyelembe vételével. Mindehhez remek eszközként rendelkezésemre áll majd a **VPlanet** környezete, mellyel számos problémát leszek képes szimulálni, amennyiben szükség lesz rá. Ehhez természetesen elengedhetetlen lesz újabb ismeretek szerzése, valamint a szerzők által publikált cikk további, mélyrehatóbb tanulmányozása is. Utalva a problémák diverzitására, s a publikáció komplex felépítésére, több évnyi kutatást és tudásbővítést képes biztosítani a szóban forgó irat. Bízom benne, hogy a következő években lesz alkalmam több problémát is tanulmányozni, továbbá a rájuk alkotott szimulációkat és számítási módszereket megismerni, az egyes hivatkozott cikkekből pedig valamennyit átolvasni.

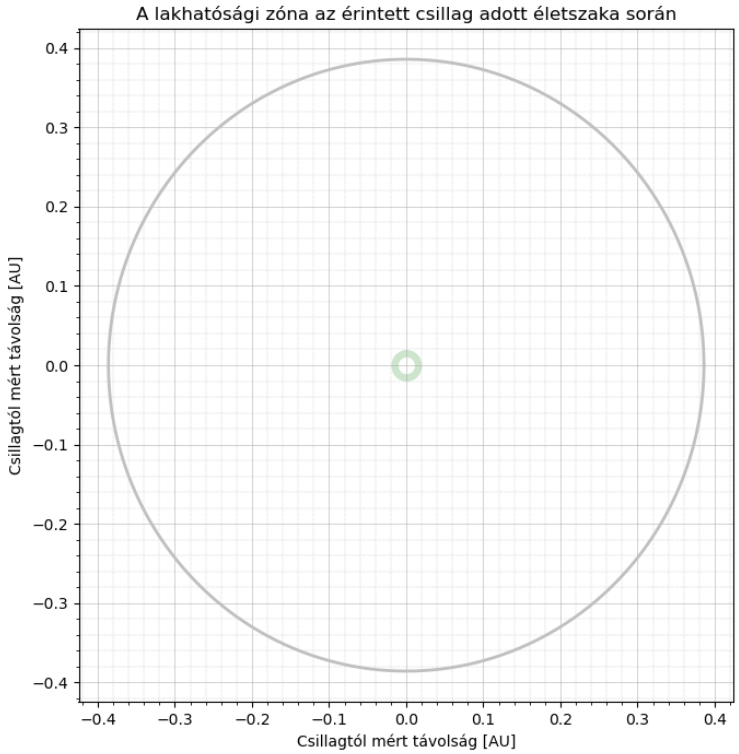
7. Köszönetnyilvánítás

Ebben a néhány sorban szeretnék köszönetet mondani azoknak az embereknek, akik nélkül nem tartanék ott, ahol. Dolgozatom nem jöhetett volna létre a témavezetőm, Dr. Szalai Tamás, tudományos munkatárs rendszeres ellenőrzése és tanácsai nélkül, aki már az alapképzés óta mindvégig segített nekem megtalálni azt az utat, amit követhetek és szívesen is egyengetek. Emellett kollektíven megköszönöm az odaadó figyelmet és biztatást valamennyi családtagomnak és barátomnak, akikre bármikor számíthattam, amikor csak szükség volt rájuk. Nem utolsó sorban pedig kollégáimnak, szaktársaimnak is köszönöm a kitartást, s hogy együtt érhettünk el eme könnyűnek nem mondható, mégis kihívásokkal és érdekességekkel teli képzés végére.

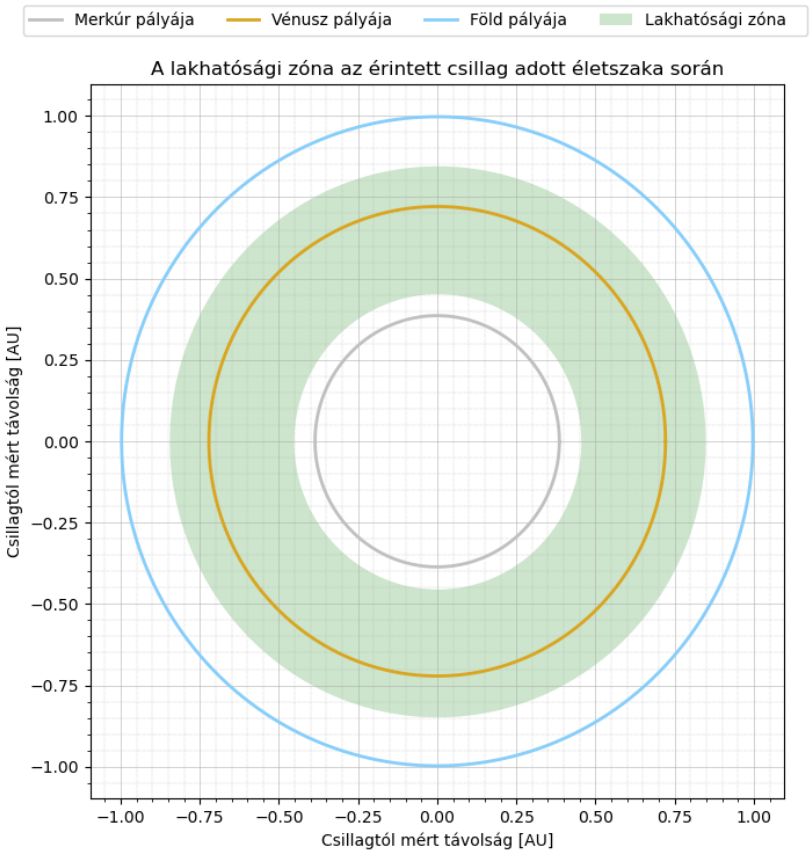
8. Függelék

Kor[Gyr]	Sugár[km]	T_{eff}[K]
1	620000	5780
2	645000	5705
3	661000	5750
4	682000	5760
5	706000	5800
6	734000	5820
7	769000	5820
8	811000	5780
9	870000	5750
10	957000	5705

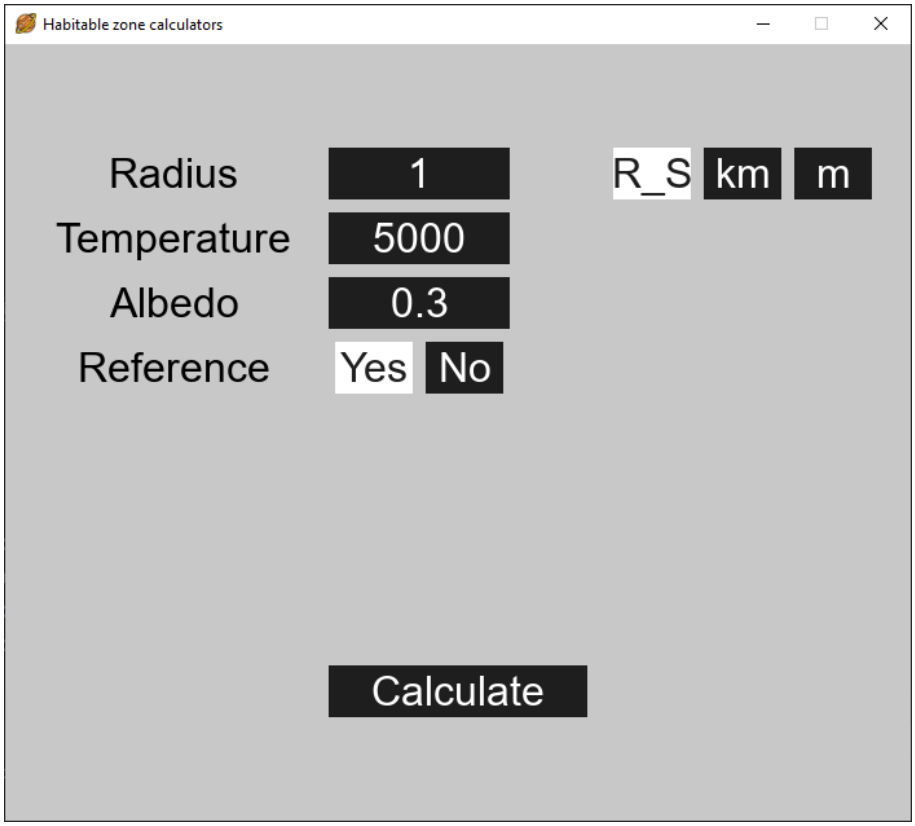
3. táblázat. A becsült sugár- és hőmérsékletértékek Napunk fejlődése során.



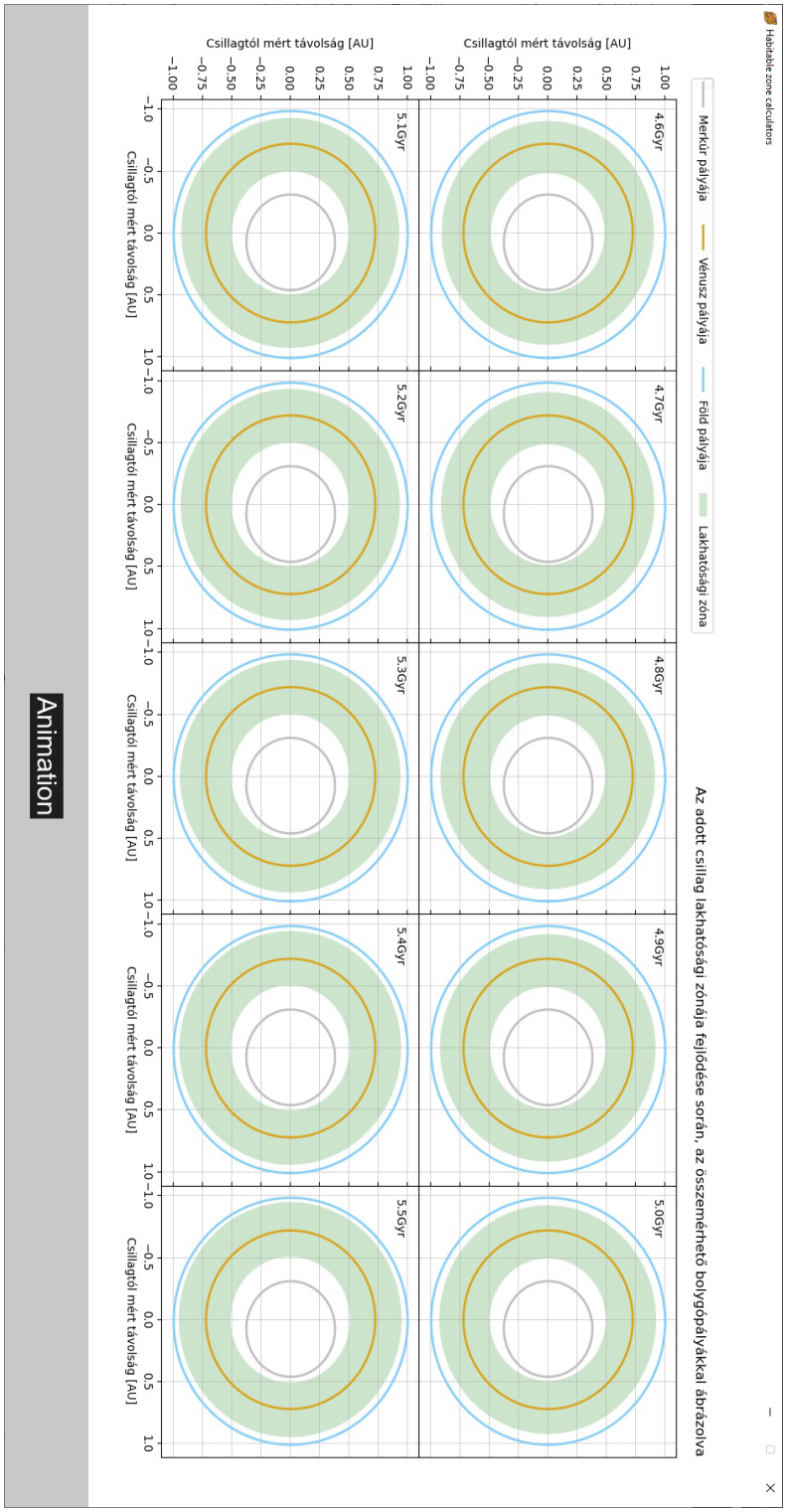
8.1. ábra. A szoftveremmel ábrázolt TRAPPIST-1 rendszer lakhatósági zónája (zöld), összevetve a Merkúr pályájával (szürke).



8.2. ábra. Az új külsőt kapott program HZ esetre nézve.



8.3. ábra. A HZ menü beviteli lehetőségei. Az eddigi működéshez hasonlóan továbbra sincs szükség fájlbemenetre.



8.4. ábra. A felújított EHZ mód statikus, teljes ábrája, működés közben.

Hivatkozások

[1] I. Ribas (2009) *The Sun and stars as the primary energy input in planetary atmospheres*, Solar and Stellar Variability: Impact on Earth and Planets, Proceedings of the International Astronomical Union, IAU Symposium, Volume 264, p. 3-18, fig. 1

[2] Rory Barnes et. al (2019) *VPlanet: The Virtual Planet Simulator*, Publications of the Astronomical Society of the Pacific, Volume 132, Number 1008

[3] http://simbad.u-strasbg.fr/simbad/sim-basic?Ident=2MASS+J23062928-0502285

[4] http://simbad.u-strasbg.fr/simbad/sim-id?Ident=Kepler-36

[5] E. Lopez, J. Fortney (2013) *The Role of Core Mass in Controlling Evaporation: the Kepler Radius Distribution and the Kepler-36 Density Dichotomy*, The Astrophysical Journal, Volume 776, Issue 1, article id. 2, 11 pp.

[6] https://virtualplanetarylaboratory.github.io/vplanet/install.html

[7] https://github.com/VirtualPlanetaryLaboratory/vplanet

[8] https://virtualplanetarylaboratory.github.io/vplanet/help.html#input-options

[9] I. Baraffe, D. Homeier, F. Allard, G. Chabrier (2015) *New evolutionary models for pre-main sequence and main sequence low-mass stars down to the hydrogen-burning limit*, Astronomy Astrophysics, Volume 577, id.A42, 6 pp

[10] D. Segransan, P. Kervella, T. Forveille, D. Queloz (2003) *First radius measurements of very low mass stars with the VLTI*, Astronomy and Astrophysics, v.397, p.L5-L8

[11] P. Kervella, F. Thévenin, C. Lovis (2016) *Proxima's orbit around Alpha Centauri*, Astronomy Astrophysics, Volume 598, id.L7, 7 pp.

[12] Guillem A. et. al (2016) *A terrestrial planet candidate in a temperate orbit around Proxima Centauri*, Nature, Volume 536, Issue 7617, pp. 437-440

[13] https://docs.google.com/document/d/1dEKcKgNX2ZRzo6qdTW9x5BloSqEalIkb7kEY_lY5lQs/edit?usp=sharing

Nyilatkozat

Alulírott Murvai Adrián Csaba, Csillagász MSc szakos hallgató (CTT50T), *Bolygórendszerek paraméter- és lakhatóságizóna-evolúciójának vizsgálata Python-szimulációk segítségével* című diplomamunkám szerzője fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló munkám eredménye, saját szellemi termékem, abban a hivatkozások és idézések általános szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Szeged, 2021. év 05. hó 22. nap

aláírás