**STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST**

**Obor č. 18-20-M/01: Informační technologie**

**PlanetLab 2 – výuková vesmírná průzkumná hra**

**PlanetLab 2 - an educational space exploration game**

**Autoři:** Ondřej Hývnar, Adam Krása

**Škola: Gymnázium a Střední průmyslová škola elektrotechniky a informatiky, Frenštát pod Radhoštěm, příspěvková organizace, Křižíkova 1258, Frenštát pod Radhoštěm, 744 01**

**Kraj:** Moravskoslezský kraj

**Konzultant:** Mgr. Richard Štěpán, Ing. Lubomír Spáčil

Ve Frenštátě p. Radhoštěm, 09. 02. 2019

**Prohlášení**

Prohlašuji, že jsme svou práci SOČ vypracovali samostatně a použili jsme pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašujeme, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemáme závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

Ve Frenštátě p. Radhoštěm, dne 09.02.2019 ………………………………………………

Ondřej Hývnar, Adam Krása

**Poděkování**

Rád bych poděkoval panu učiteli Richardovi Štěpánovi za to, že nám dovolil pracovat v týmu a také, že nám dovolil věnovat se svým projektům v hodinách.

**Anotace**

Ve své práci SOČ jsme se zabývali vytvořením výukové vesmírné hry. Původní nápad byl založen na C++, kde jsme si vytvořili vlastní engine v Direct3D, avšak C++ bylo pro naše účely hodně limitované a bylo složité implementovat některé funkce, které jsme potřebovali, takže jsme hru předělali v C# na engine Unity3D. Většina obsahu, který se ve hře nachází je buďto námi vytvořená (modely, textury, hudba), použitý z defaultních Unity assetů nebo z volných zdrojů („non-commercial use repositories“). Textury planet jsme stáhli ze stránky, která je uvedena v zdrojích/odkazech.

**Klíčová slova**

Vesmír, průzkum, PlanetLab, PlanetLab2, C++, C#, Unity3D, planety, vesmírná loď, Direct3D

**Annotation**

In our SOČ work we have created an educational space exploration game. The original idea was made on C++, where we have made our own engine in Direct3D, but C++ was very limited for our uses and was very hard to implement certain mechanics, so we had rewritten the game in C# and ported it to the Unity3D engine. The assets in this game are either self-made (models, textures, music/sounds), used from default Unity assets or used from non-commercial use repositories.

**Keywords**

Space, exploration, PlanetLab, PlanetLab2, C++, C#, Unity3D, planets, spaceship, Direct3D

**Úvod**

PlanetLab2 je sandboxová(hráč není limitován v tom, co může dělat) výuková hra, ve které hráč ovládá vesmírnou loď, se kterou prozkoumává Sluneční soustavu. Pokud chce hráč získat informace o určité planetě, musí ji naskenovat, díky čemuž se mu zobrazí v panelu informace o povrchu, poloměru, váze, hustotě, tlaku atmosféry a poměru křemíku/vodíku/železa.

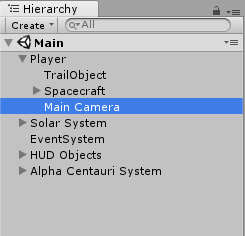
**Ovládání**

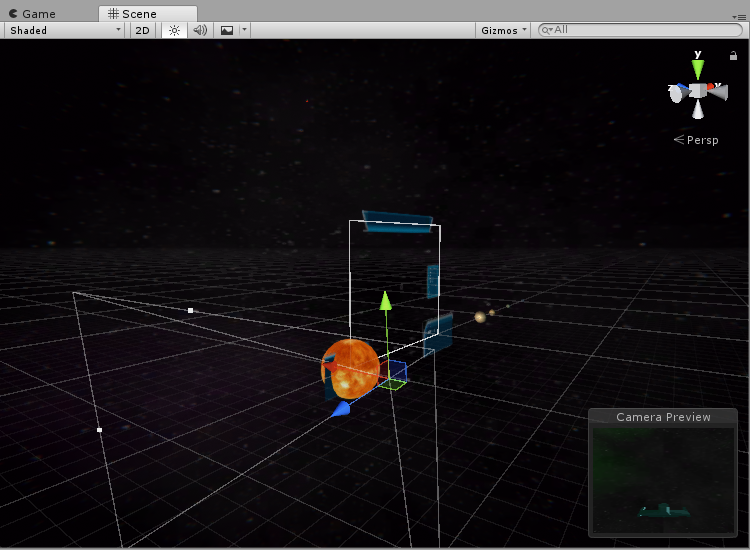
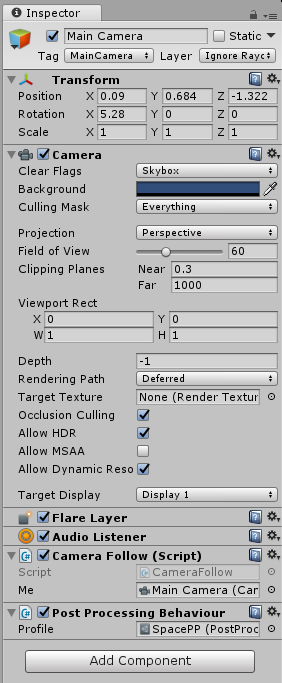
**W/S** - Ovládání zrychlení

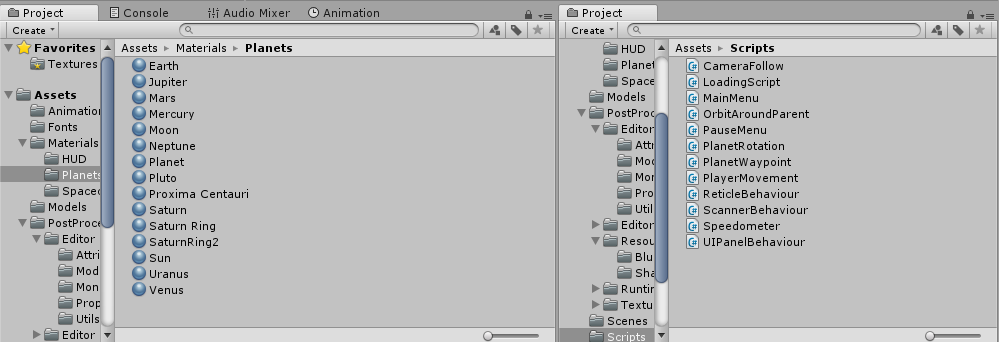
**R/F** - Ovládání náklonu do výšky

**A/D** - Ovládání horizontálního náklonu

**Q/E** - Ovládání vertikálního natočení

**Jak hra funguje** 

Nejprve bychom si měli představit prostředí Unity. V Unity3D se pracuje v takzvaném Inspektoru s objekty, ke kterým jde přiřazovat vlastnosti a skripty. Na obrázku můžeme vidět seznam všech objektů. Funguje zde hierarchický systém, tzn. objekty lze přiřazovat pod jiné objekty (parent objects & child objects). Dědičnost se může vrstvit do nespočetna.  
  
Dalším panelem je panel náhledu, kde můžeme vidět herní prostor v režimu editace scény nebo v herním režimu, kdy se hra “debuguje”.

Taktéž se zde nachází i Project View jako u Visual Studia.

Dále se tedy dostáváme s onomu Inspektoru, který nám dovoluje přidávat a měnit vlastnosti, tzv. komponenty.

Zde máme otevřený objekt Main Camera, což je objekt, který nám umožňuje vidět dění ve hře. Každý objekt má svou vlastnost “Transform”, která určuje jeho pozici na ploše, absolutní rotaci (vzhledem k XYZ ose) a jeho velikost/měřítko. Nehmotné objekty se chovají jako body a nemají rozměry, tzn. hodnota scale u nich nijak nepůsobí.  
  
Následně vidíme komponentu s názvem Camera, která nám převádí nastavení kódu kamery na hezký GUI “formulář”, tzn. nemusíme přepisovat kód, abychom změnili jednu hodnotu, takto si ji můžeme pohodlně změnit přímo z editoru.  
  
Ve vlastním kódu můžeme tyto hodnoty označit tagem SerializeField a následně se nám ukážou v Inspektoru. Taktéž můžeme přidat tag Range a tím místo zadání hodnoty můžeme v Editoru hýbat “šoupátkem”.  
  
Konkrétní příklady:

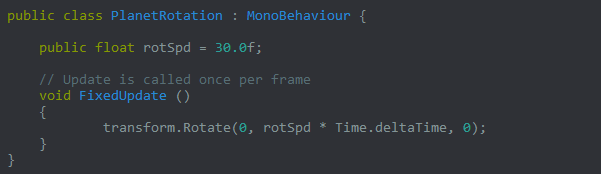
> bez tagu Range  

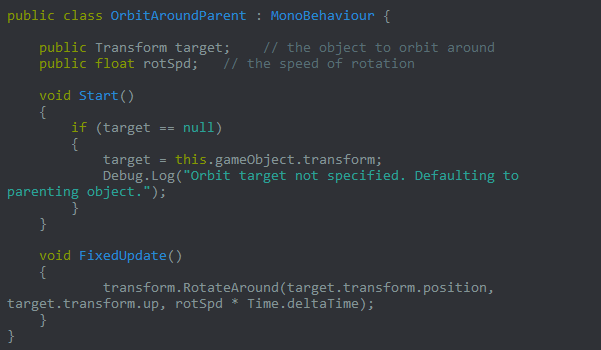

> s tagem Range  

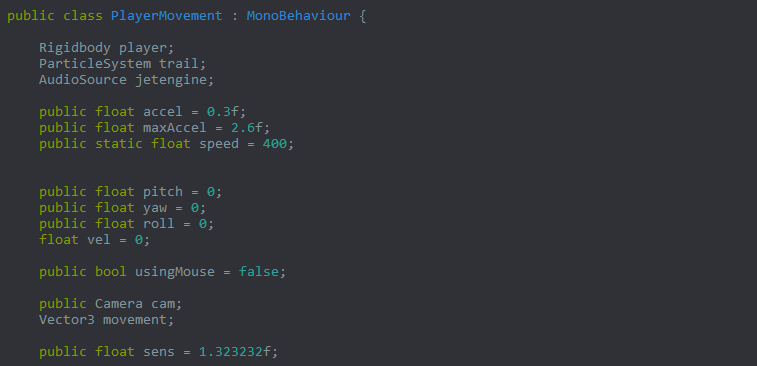
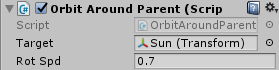

V nastavení kamery se nastavuje také Skybox, v našem případě tedy spíš “okolní hvězdný obzor”Box, protože pomocí Skyboxu můžeme vykreslit objekty, které jsou pro kameru v Inf vzdálenosti. Naše kamera totiž po malých úpravách funguje stejně jako by to byla reálná kamera. Vidí hloubku, odlesky, má ohniskovou vzdálenost, dobu expozice, má na čočce prach a malé oděrky apod.  
Jak můžeme vidět na přiloženém obrázku, kamera vidí rozmazaně, má svou přirozenou ostrost a jde vidět “špína” na čočce. Takto nám to přišlo reálnější, než zobrazovat svět “na sucho” a prázdně. Dodává to malou špetku pocitu “uvěřitelnosti”.

**Generování planet a jiných vesmírných těles**

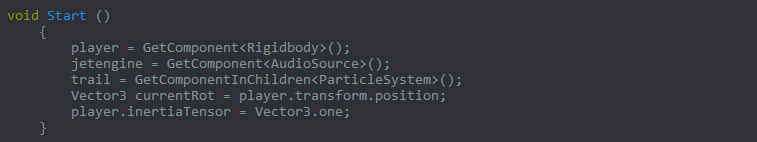
Planety jsou staticky nastaveny na své pozice, avšak jejich rotace je řízena funkcí v kódu.

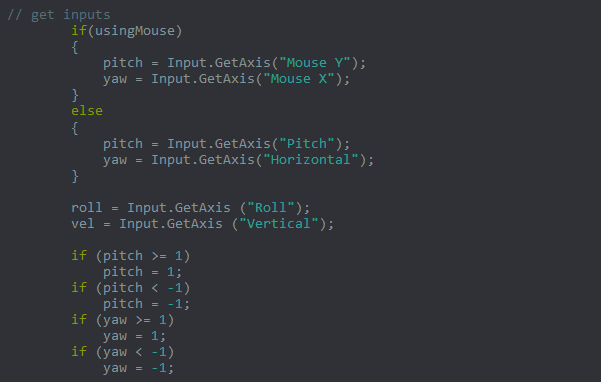
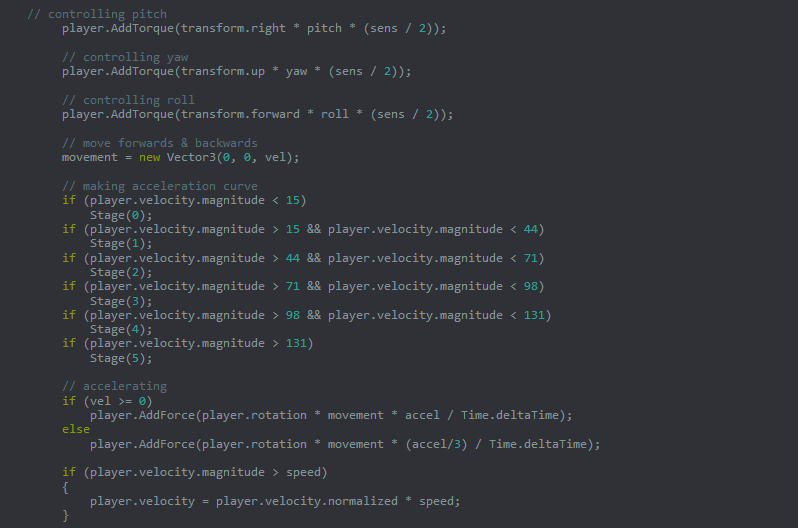
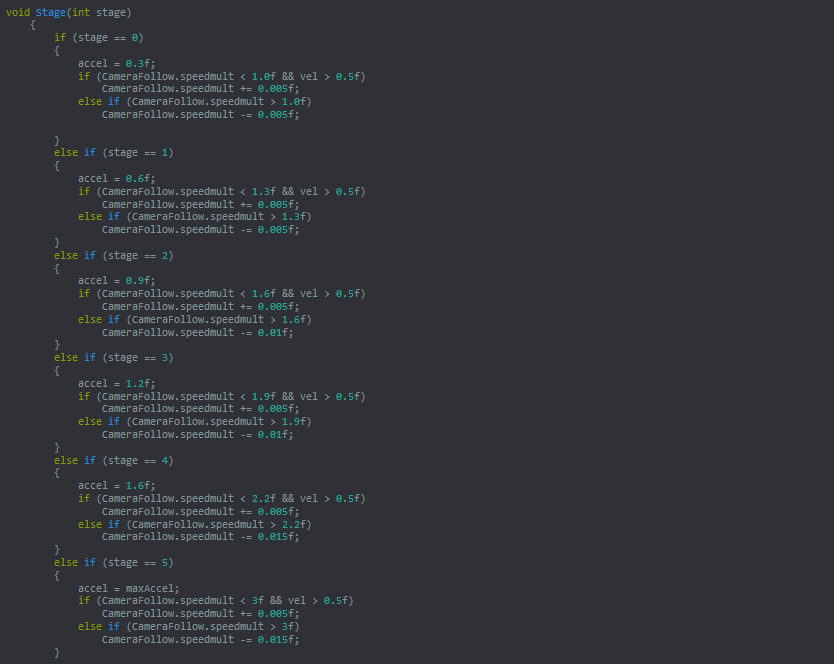
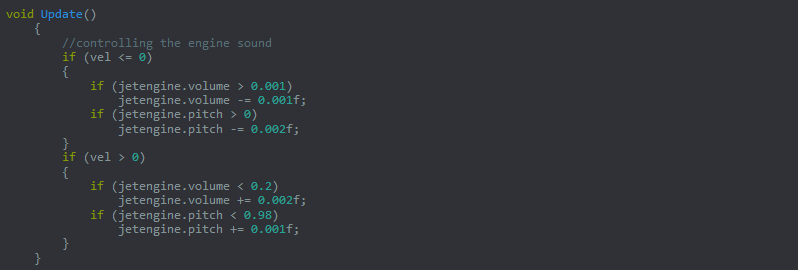


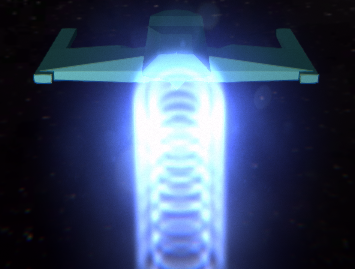
Planety rotují kolem své osy (defaultně rychlostí 30 stupňů za tick, což nakonec vychází neskutečně rychle, ale u každé planety jsou jejich přibližně reálné hodnoty, takto aspoň uvidím, kde jsem hodnotu nepřidělil). Následně můžeme vidět funkci FixedUpdate(), která se provádí každý tick. Na rozdíl od funkce Update(), která se provádí každý frame, funkce FixedUpdate počítá s lagy/záseky, takže při nízkém počtu snímků za vteřinu dopočítává svůj kód v “reálném čase”, tzn. i když se nám bude sekat hra, tak se nestane, že by se nám před očima stačila planeta otočit rychleji než by měla, což by znamenalo skok v obraze nebo naopak při nízkém počtu snímků se točila pomaleji a při vysokém počtu snímků zase rychleji. Veškeré výpočty, které potřebují přesnost, jako třeba herní fyzika nebo pohyby se doporučují posílat do FixedUpdate fronty. Taktéž existuje i funkce LateUpdate(), která má nejnižší prioritu a vykonává se po každém vykresleném snímku. Do této řady se přidává například postprocessing a jiné “low-priority tasks”.  
  
Na následujícím kusu kódu můžeme vidět, jak funguje orbitování planet okolo jejich “rodičovské planety”. Berme v potaz to, že pro zjednodušení se i hvězdy, jako například naše Slunce, počítají jako planety.  
  
Zde je taktéž vidět rotace ve funkci FixedUpdate. Taktéž je zde vidět funkce Start(), která by šla nahradit akcí OnAwake(), ale jelikož nám stačí tato funkce zavolat pouze jednou při spuštění tohoto objektu, tak nám plně postačí start, kdy se zkontroluje, zda target není prázdný.

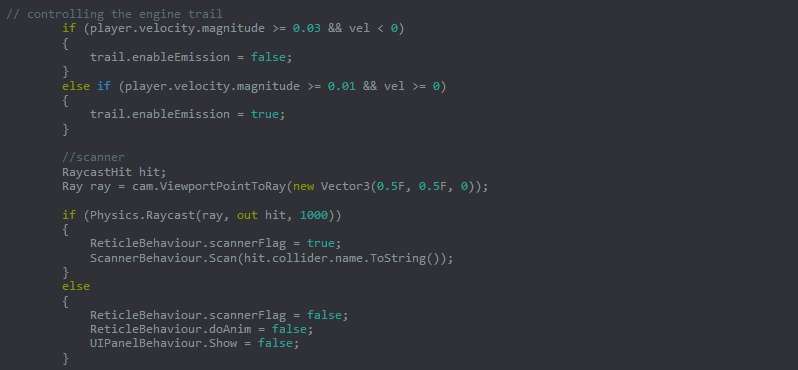
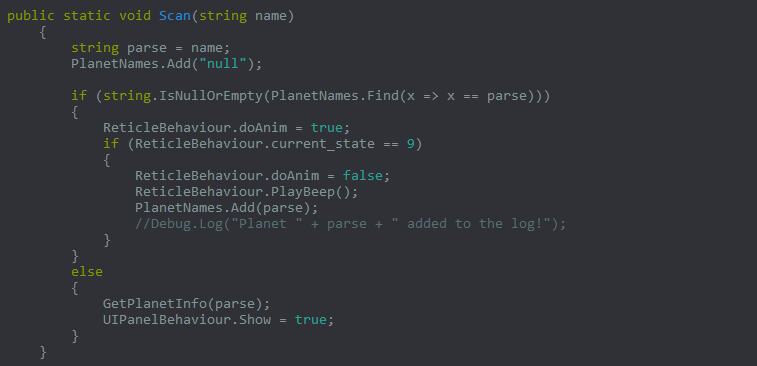
****Jak jste si mohli povšimnout, tak v kódu není tag SerializeField a přesto se nám hodnota zobrazuje v Inspektoru. Toho jsme dosáhli tak, že proměnné s parametrem “public” se automaticky přiřazují do Inspektoru. Taktéž je popíše podle CamelCase normy.  
  
  
**Pohyb hráče**  
V následujících částech kódu můžeme vidět, jak funguje pohyb hráče.

Zde můžeme vidět, jak se komponenty zaevidují jako odkazy na objekty. Pomocí funkce GetComponent se program odkazuje na objekty z Inspektoru. Jak můžeme například vidět GetComponentInChildren, která vybírá sama komponenty i z objektů potomků.



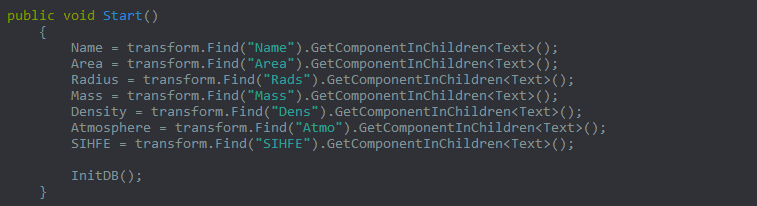
V následující části můžeme vidět, jak se zpracovávají vstupy z os. Máme proměnné pro každou osu pohybu, které přiděluje hodnotu z každé „controllerové“ osy. Osa nabývá hodnot -1 až +1. Je zde přepínač mezi použitím inputu z myši a z klávesnice. Taktéž je zde „pojistka“ proti přetížení osy.  V další části kódu lze vidět, jak se vyhodnocují data z ovládacích os a také i řešení zrychlování přes takzvané „Stages“ neboli stupně, které si popíšeme na dalším obrázku. **** Stupně se spouští při dosažení dané rychlosti lodi, kdy se sepne vždy další stupeň zrychlení. ****Následně po vyhodnocení stupně se aplikuje zrychlení a pokud rychlost přesáhne maximální rychlost, tak se rychlost normalizuje, aby nedošlo ke škubnutí.  
Zde si tedy ukážeme jak a co nastavují akcelerační stupně. Jde totiž o jednoduchý if-else strom, kdy se projede, který stupeň se má nastavit. Poté se upraví akcelerace a zorné pole kamery.  
  
Nastavení kamery si ukážeme později, nyní se podíváme ještě na ovládání grafických a zvukových efektů tryskového motoru.Zvuk je ovládaný pomocí rychlosti plavidla.****

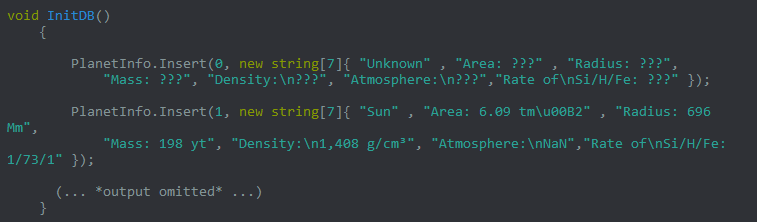
Efekt motoru se tvoří několika za sebou jdoucími svítícími kruhy, které po složení tvoří efekt trysky. ****

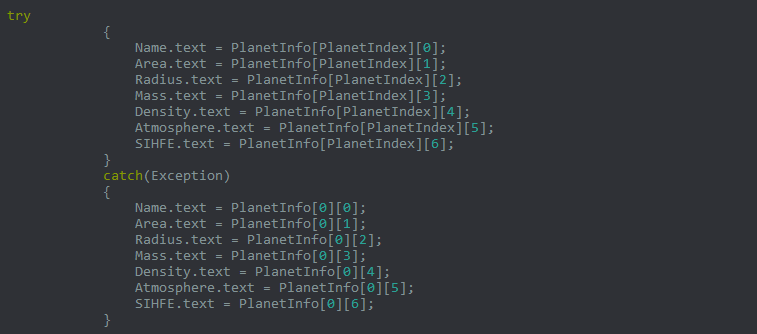
****Taktéž je zde základní kámen skeneru. Což je vytvoření RayCastu, paprsku, který putuje prostorem, dokud nenarazí na objekt bez flagu IgnoreRaycasts. Loď neustále vytváří paprsky, které kontrolují, zda něco není v hledáčku. Poté přijde na řadu metoda Scan. ****

Metoda Scan projde databázi a zjistí, jestli je planeta známá nebo ne. Pokud ne, najde si info v globální databázi a uloží si ji do lokální databáze.

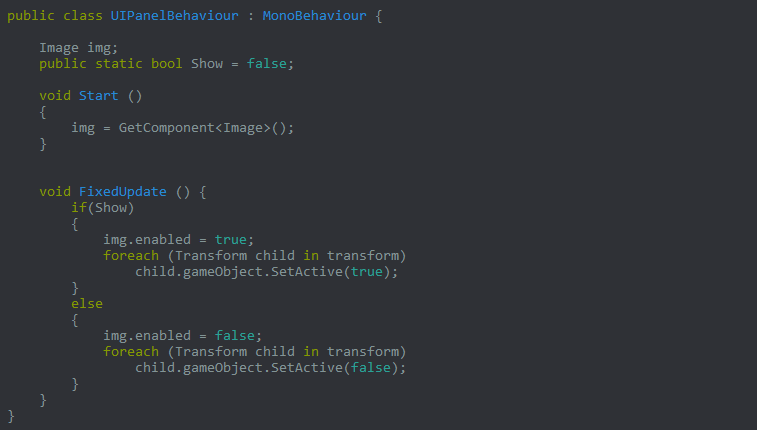
****

Prvně se musí nastavit odkazy a zavést databáze. ****Ve Startu se spustí InitDB, která vytvoří záznamy pro všechny známé planety.

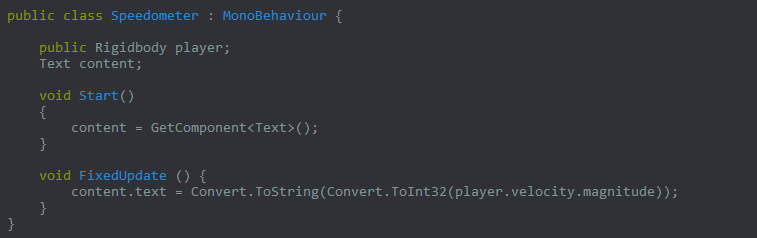
****

Po úspěšném skenu následuje zkouška zapsání výsledku do paměti. Pokud se něco pokazí, nahraje se defaultní záznam.****

Po doskenování vyskočí ve hře tabulka s oskenovanými hodnotami. Ta je řešena pomocí třídy UIPanel. Její kód můžeme vidět níže.

****

Také zde nesmí chybět jednoduchý ukazatel rychlosti.

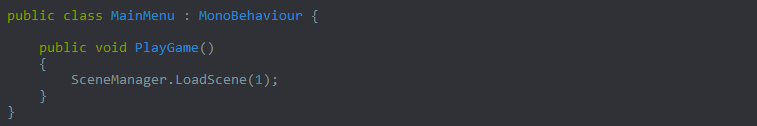
****

Taktéž hra obsahuje „waypointy“ nebo-li záchytné body, které jsou přimknuté na hlavní hvězdu dané soustavy.



**Hlavní menu, pauza a načítací obrazovka**

Každý správný program by měl mít úvodní obrazovku a tenhle není výjimkou. Hlavní menu je velice jednoduché a skript k němu ještě jednodušší.

****

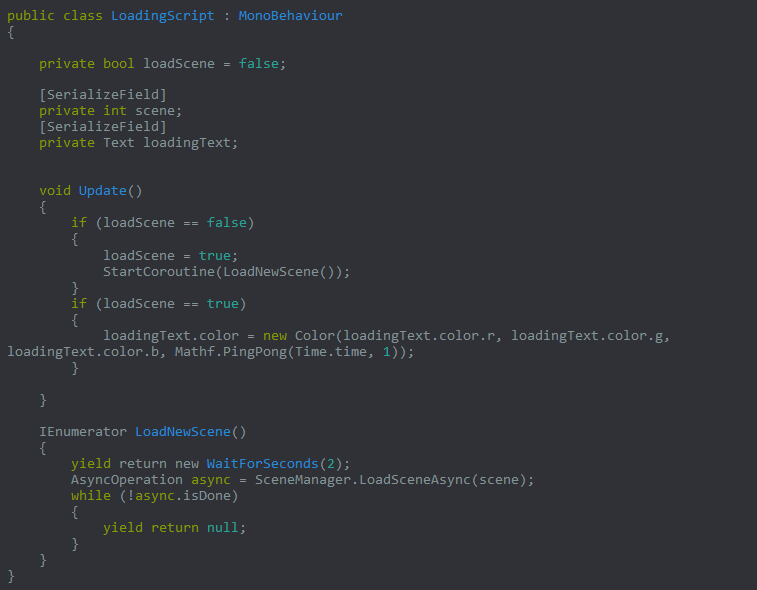
Hlavní menu obsahuje pouze logo hry, 3D pozadí a tlačítko „Play“, které po stisknutí načte Scénu 1.

****

Zde na scénu přichází načítací obrazovka, která je taktéž velice jednoduchá. V pozadí rotuje Slunce a dole problikává nápis Loading.

****

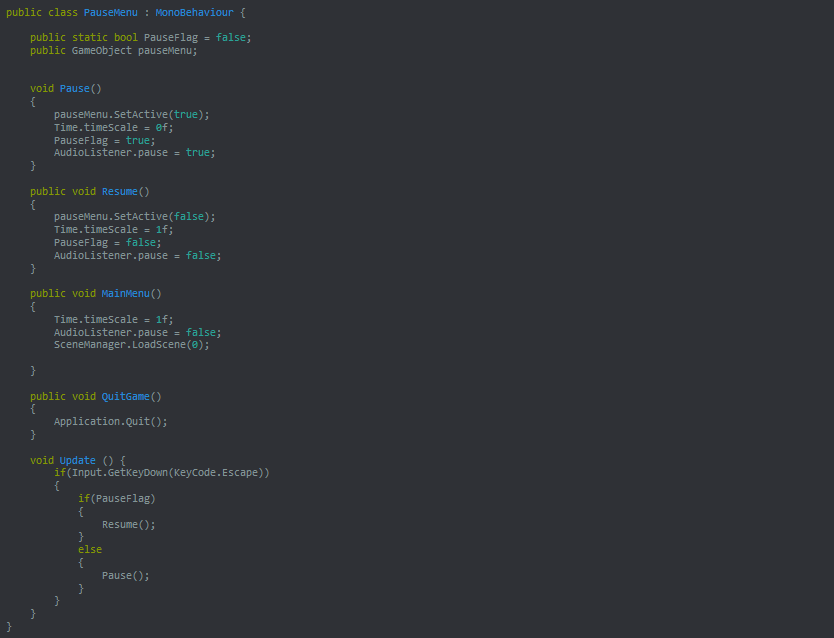
Načítání probíhá díky volání ko-rutiny, což je vlastně podproces, který pracuje paralelně vůči tomuto objektu.

****

Jako poslední zmíňku zde máme pause menu, které je opět velice jednoduché.



Níže můžeme vidět jak pause menu funguje.

****

**Závěr**

Při tvoření této aplikace jsme prohloubili naše zkušenosti s enginem Unity3D a programovacím jazykem C#. Taktéž jsme si procvičili spolupráci a komunikaci.

**Zdroje**

[**https://unity3d.com/**](https://unity3d.com/)[**https://github.com/SakDrakken/PlanetLab2**](https://github.com/SakDrakken/PlanetLab2)

**https://www.solarsystemscope.com/textures/**