**Inside the matrix**

**Autori: Olariu Ciprian**

**Cernat Catalin**

**Ganea Dan**

**Hareza Andrei**

**Despre Inside the matrix**

Inside the matrix este un joc de tip labirint structurat pe nivele de dificultate ce dezvolta memoria fotografica a jucatorului.

Interfata este una simplista si animata, personajul principal fiind Link din Zelda si elementele de decor fiind de natura Dungeon/Castle.



Jocul consta in a controla personajul, aflat initial in mijlocul incaperii, cu scopul de a ajunge la unica iesire din labirint intr-un timp cat mai scurt.

Jucatorul poate alege intre trei nivele de dificultate, ce afecteaza dimensiunea incaperii si prin urmare timpul necesar evadarii. La fiecare joc nou, se va genera un labirint random, cu dimensiunile conforme nivelului ales.

**Elemente jocului**

Jocul este creat in Unity 5.3.4, scripturile fiind realizate in C#.

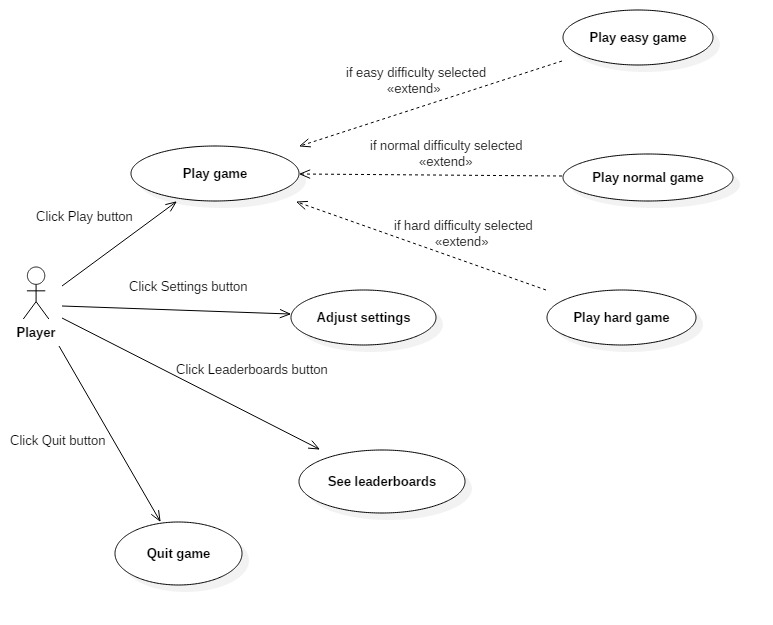
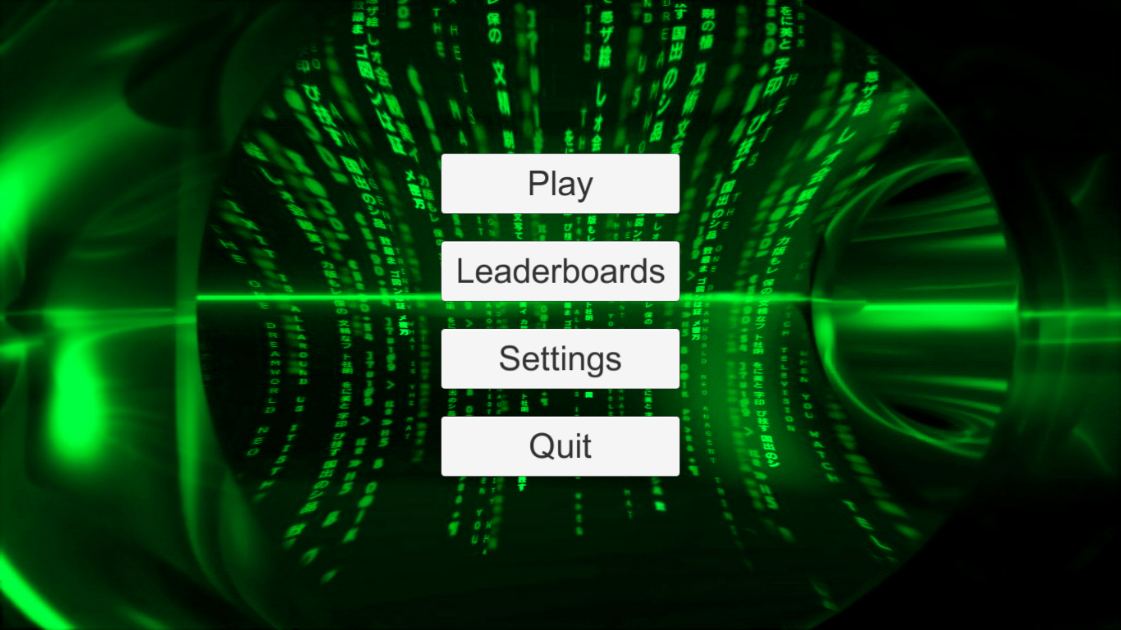


Diagrama de mai sus prezinta cazurile de utilizare ale jucatorului aflat in meniul jocului, ce contine:

* **Play Game** ( -> care te directioneaza in joc dupa ce selectezi dificultatea );
* **Settings** ( -> setarile aplicatiei);
* **Leaderboards** ( -> clasamentul scorurilor obtinute pana acum );
* **Exit** ( -> buton ce permite iesirea din joc );



In Settings si Leaderboards avem cate un buton de **Back** care ne directioneaza catre meniul principal.



Aceste directionari din meniul principal in setari sau din setari in meniul principal au fost facute usor cu ajutorul Unity adaugand nivelele folosite, mai precis adaugand scriptul SceneChanger.cs pentru fiecare scena, iar aici precizam functia LoadScene din implementare prin care trimitem numele scenei ( MainMenu, Leaderboards sau Settings ).

public void LoadScene(string sceneName)

{

SceneManager.LoadScene(sceneName);

}

Jocul propriu-zis este format din mai multe scripturi C#:

* CameraMove
* CharacterAnimator
* CharacaterMove
* DifficultyManager
* GameOver
* MazeExit
* MazeGenerator
* SceneChanger
* Timer

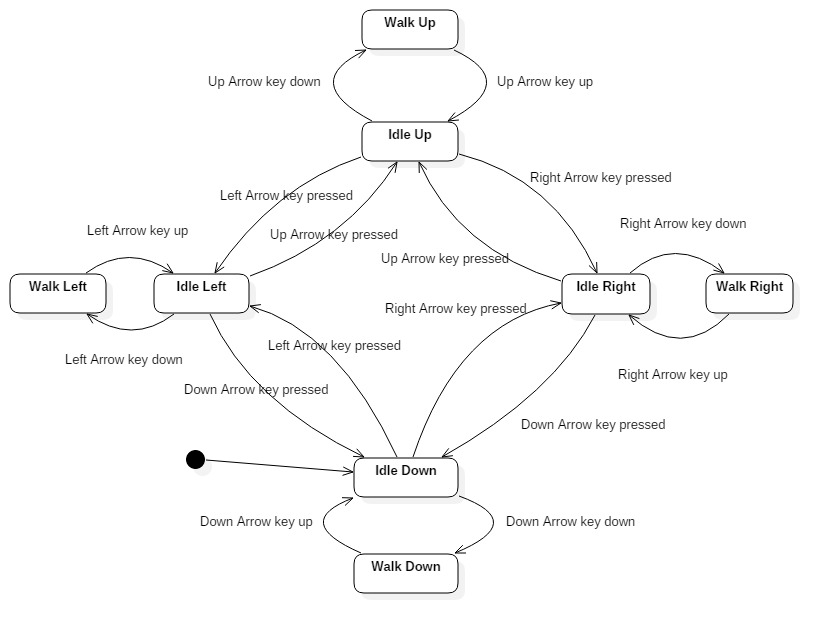
**CameraMove**

-la initializarea scriptului in timpul jocului se ia o referinta catre personaj, iar la fiecare frame se muta camera astfel incat sa fie centrata pe pozitia personajului

**CharacterAnimator**

-prefab-ul personajului din joc are atasat un animator care realizeaza animatiile miscarii lui in cele patru directii

-in acest script, la actionarea tastaturii se seteaza niste variabile booleene ce declanseaza mai departe tranzitii in masina de stari a animatorului, conform diagramei de stari de mai jos



**CharacterMove**

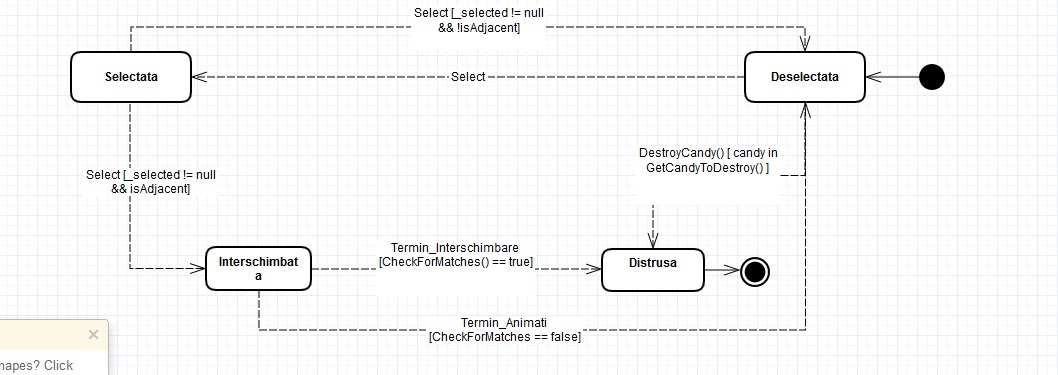
-la fiecare frame, in functie de tastele apasate, personajului i se aplica o forta ce il misca intr-una din cele patru directii

-cu ajutorul componentelor Unity RigidBody2D si BoxCollider2D aplicate pe personaj si pe zidurile din labirint, personajul nu poate trece prin zidurile cu care face coliziune in timpul deplasarii

**Candy**

-stocheaza informatii despre fiecare bomboana in parte: spriteul asociat si tipul ales dintr-un Enum:

*public enum CandyType {Croissant, Cupcake, Danish, Donut, Macaroon, SugarCookie}*



**CandyMatrix**

Clasa are urmatoarele atributii:

1. Crearea matricii

Se realizeaza prin intermendiul functiei *CreateMatrix() c*e genereaza random o matrice de dimensiune \_rows x \_columns testand la fiecare pas sa nu se creeze trei bomboane de aceeasi culoare consecutive



2. Simuleaza o mutare

Clasa verifica in functia Selected faptul ca ultimele 2 bomboane selectate sunt adiacente si in caz afirmativ declanseaza corutina *Swap*() ce :

-extrage pozitiile dulciurilor

-declanseaza animatiile (candy.MoveToAnimation)

-apeleaza functia de interschimbare in matrice

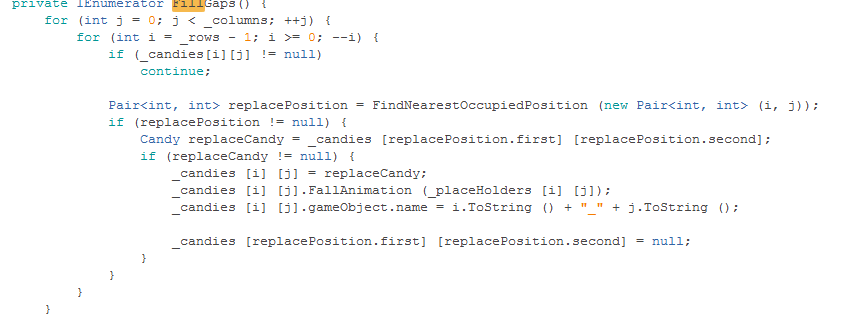
In cazul in care mutarea nu a declansat niciun 3-matching ultimii 2 pasi sunt reluati, altfel se va apela DestroyCandies(); Astfel si animatia va realiza un reverse

3. Distruge perechile formate

S-a creat o functie ***private List<Candy> GetCandiesToDistroy(Candy candy)*** care sa returneze o lista cu toate bomboanele ce trebuie distruse la pasul curent. Ea foloseste alte 2 metode auxiliare: *private List<Candy> GetHorizontalMatches(Candy candy) s*i ***private List<Candy> GetVerticalMatches(Candy candy)***ce itereaza succesiv la stanga/dreapta respective sus-jos atata timp cat avem acelasi tip de bomboana si introducem in lista daca numarul de bomboane identice nu este mai mic de 3.

Se va itera prin lista nou formata apeland functia de animatie a distrugerii ***candy.DestroyAnimation ()*** si se umplu spatiile libere in 2 pasi:

-se condenseaza spatiile ( bomboanele de sus coboara cat timp nu sunt blocate in jos)



* Se umplu random spatiile ramase sus, si se reia verificare de noi matchinguri formate reapeland corutina ***DestroyCandies ()***

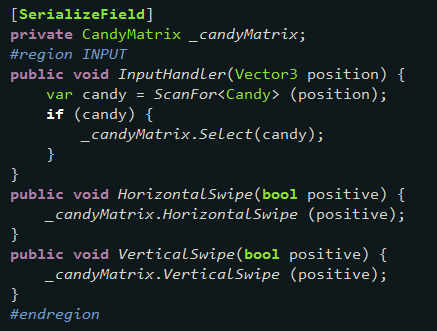
4. Creaza Shuffle

Se parcurge din nou toata matricea si sunt verificate pentru fiecare bomboana daca mutand-o in fiecare din cele 4 directi genereaza o miscare valida. Daca nu se intalneste nicio mutare valida playerul este blocat si e nevoie de un Shuffle automat:

se apeleaza corutina ***private IEnumerator Shuffle(List<Candy> candies)***  ce preia toate bomboanele din matrics si extrage random cate un element pe care il reintroduce stergandu-l din lista daca nu realizeaza matching cu cele deja existente. Daca cele ramase nu pot indeplini aceasta conditie operatia de reia. Avand in vedere ca probabilitate de shuffling este foarte mica, dupa 20 de repetitii se va renunta la idea de Shuffling regenerand matricea cu ***CreateMatrix()***

**GameManager**

Aceasta clasa este de tip **singleton** care primeste evenimente si trimite comenzi celorlalte clase mai mici.

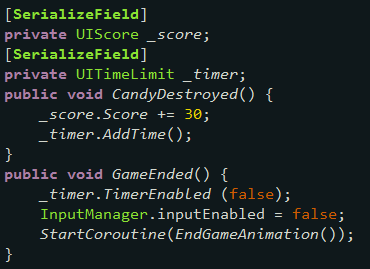
De exemplu, clasa **InputManager** ii trimite lui **GameManager** pozitia la care jucatorul a dat click, urmand ca acesta sa caute un obiect de tip **Candy** la acea pozitie. Daca exista un astfel de obiect, **GameManager** va trimite o comanda de selectare a obiectului gasit catre clasa **CandyMatrix**. 

Pe langa aceste evenimente primite de la **InputManager**, **GameManager** mai primeste si evenimente de la alte clase, cum ar fi:

* **Candy** trimite un eveniment in momentul in care este distrusa.
* **UITimeLimit** trimite un eveniment in momentul in care timer-ul atinge valoarea 0.

Acesta va trata aceste evenimente trimitand comenzi catre clase mai mici:

* Catre clasa **UIScore** se trimite o comanda de incrementare a scorului
* Catre clasa **UITimeLimit** se trimite o comanda de incrementare a timpului
* In cazul in care se primeste evenimentul GameEnded, GameManager se va ocupa de distrugerea scenei de joc si instantierea scenei de sfarsit de nivel.



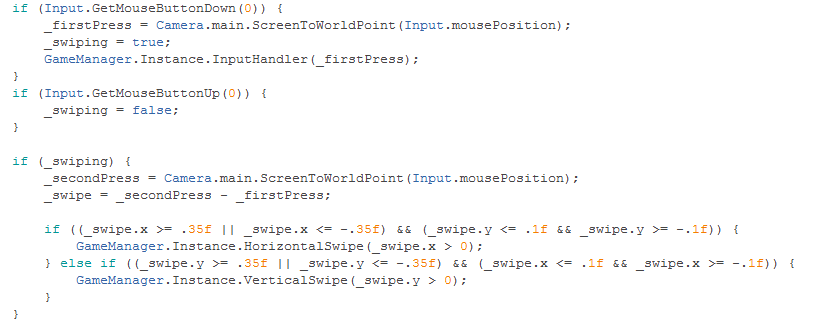
**Tween**

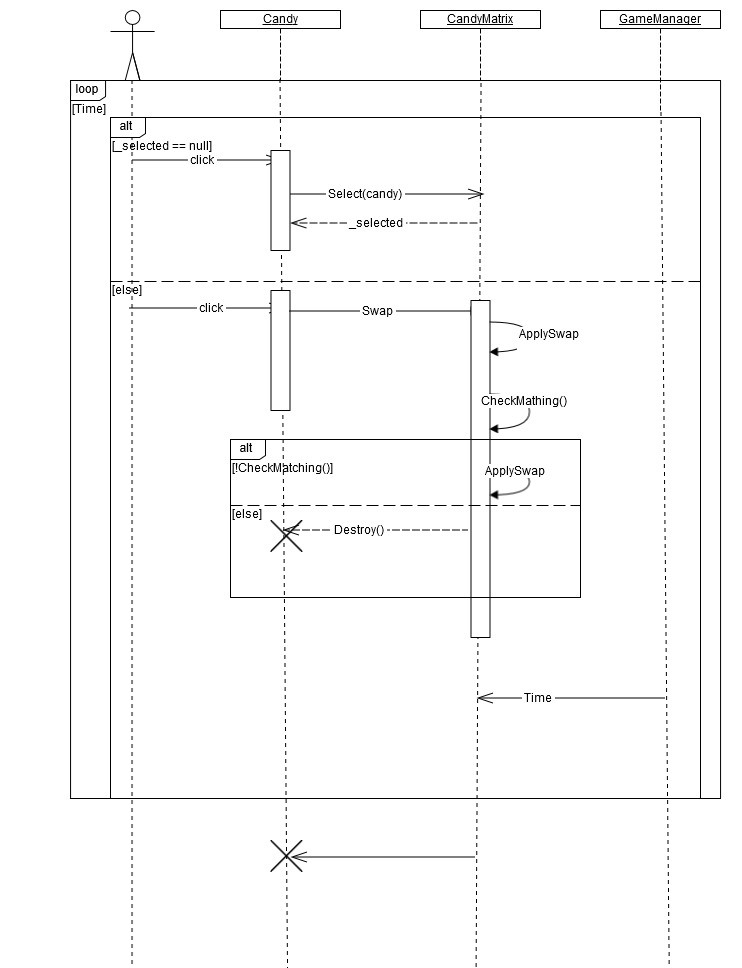
Aceasta clasa este responsabila pentru animatiile de miscare / scalare a obiectelor. De fiecare data cand este nevoie de o animatie de acest tip, aceasta clasa va crea o instanta a ei si o va atasa obiectului pe care se doreste executarea animatiei, iar acea instanta a clasei va rula animatia in functie de anumiti parametrii.

Pe langa animatiile de tip liniar, aceasta clasa suporta mai multe tipuri declarate in containerul EaseType. In functie de tipul ales, clasa va decide ce functie de interpolare va folosi in rularea animatiei.

**Input Manager**

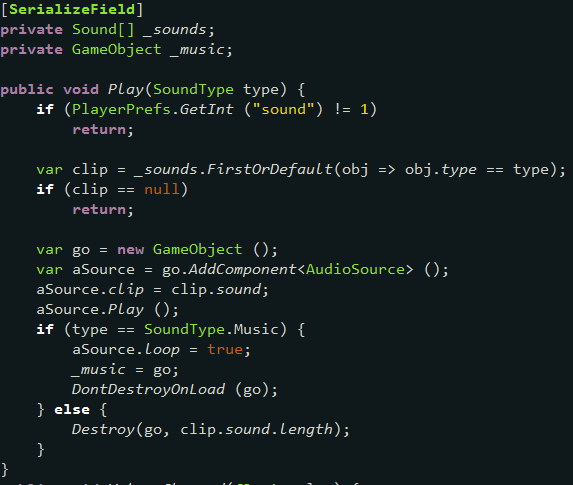
Aceasta clasa are rolul de a gestiona interactiunea cu utilizatorul, raspunsul la clickul/touchul executat de acesta. Clasa transmite GameManager-ului informatii legate de comportamentul utilizatorului, avnad grija sa blocheze accesul acestuia in timpul animatiilor.



Acest lucru este ilustrat in urmatoarea diagrama secventiala

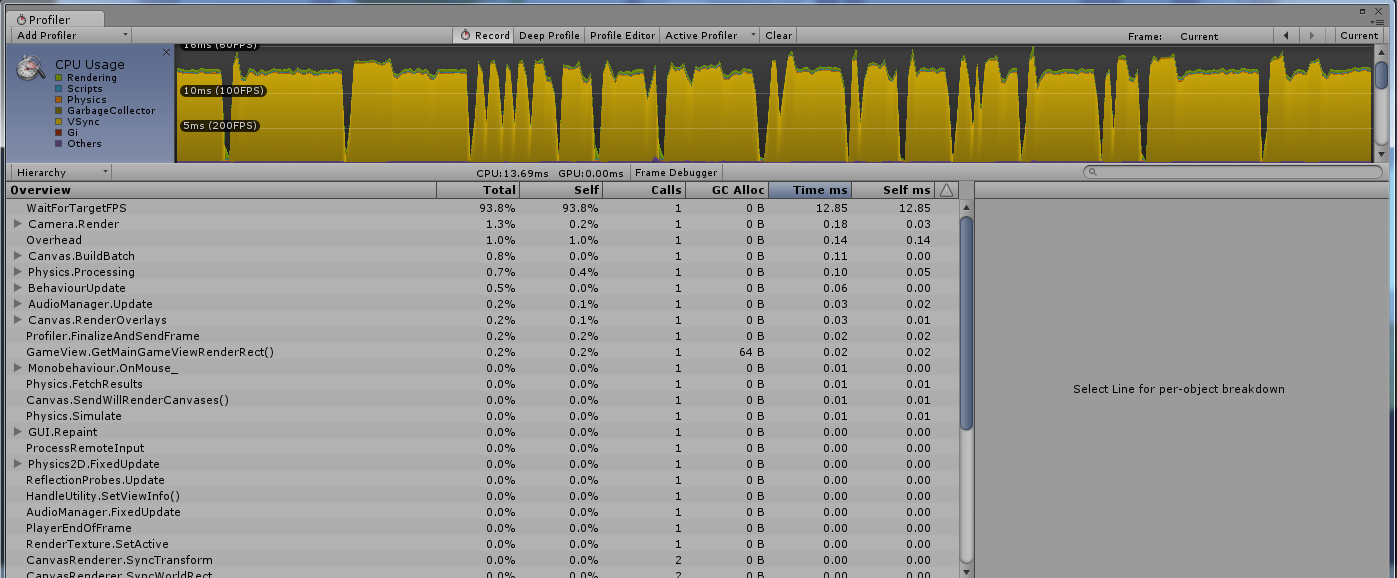
**SoundManager**

Aceasta clasa se ocupa de sunetele din joc si de setarile pentru volum. La fel ca si **GameManager**, **SoundManager** este o clasa de tip **singleton**. Aceasta clasa contine o lista de clipuri audio, fiecare clip audio avand o proprietate **SoundType**, care poate fi de mai multe feluri (Music, Select, Match). **SoundManager** poate primi comenzi de tipul PlaySound, urmate de un argument de tipul **SoundType**. Aceasta va rula un sunet in functie de tipul de sunet primit ca argument.

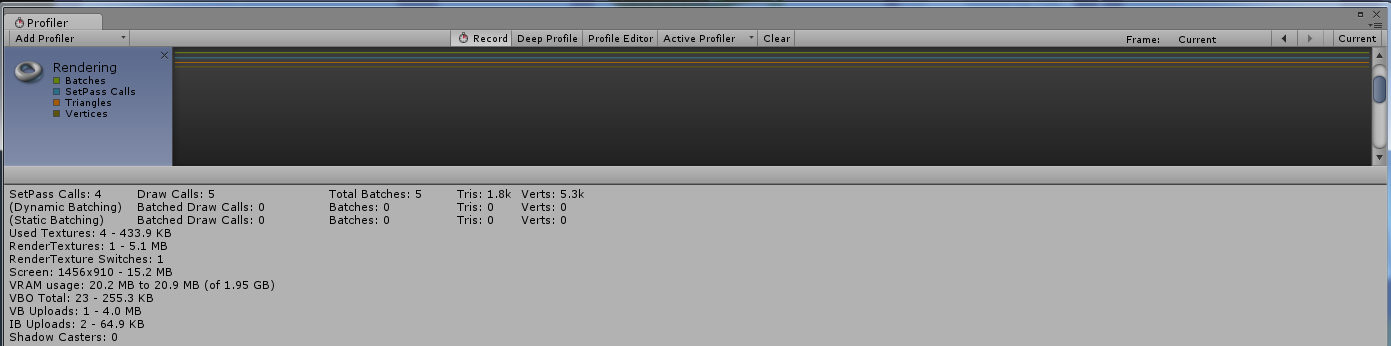


**Testarea**

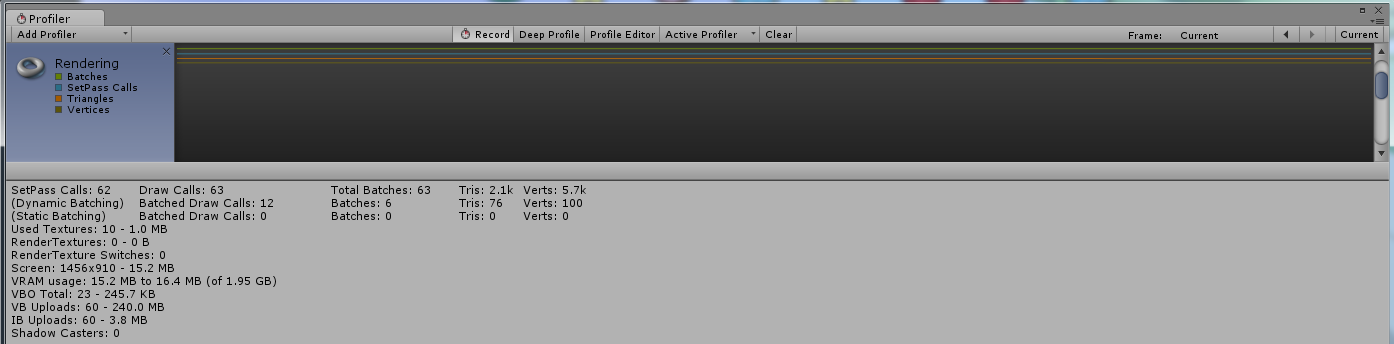
* **CPU Usage**

CPU Usage este normal, acesta aflandu-se la peste 60 frames per second majoritatea timpului. Singurul moment de “spike” se afla in momentul instantierii urmatoarei imagini de background, necesara pentru a crea animatia de miscare din meniu. Cu toate acestea, acel moment nu scade performanta CPU-ului sub 60 frames per second, aceasta valoare fiind considerata optima in game development.

* **Rendering**

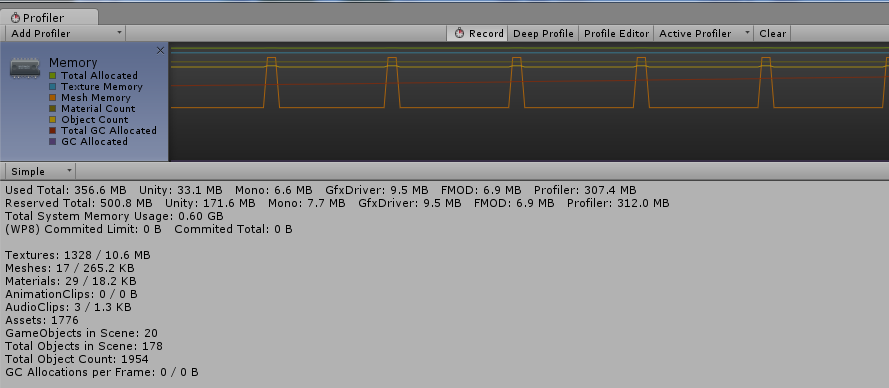
In cazul placii video, se poate observa ca pentru un ecran de 1456 x 910 memoria folosita este de aproximativ 20MB avand 5 obiecte care sunt desenate in scena de Main Menu. 

In scena de joc in schimb, memoria folosita este redusa la 15MB, iar numarul de obiecte care sunt desenate pe ecran a fost ridicat la 63. Cu toate ca numarul obiectelor desenate a crescut, observam ca memoria ramane aproximativ constanta, deoarece obiectele instantiate in scena (obiectele de tip Candy) reprezinta texturi de dimensiuni mici. Numarul de obiecte desenate poate fi redus prin crearea unei texturi mai mari care sa contina mai multe din imaginile folosite in scena. Acest lucru nu este necesar in momentul de fata.



* **Memory**

Din punct de vedere al memoriei consumate, aceasta se afla undeva in jurul valorii de 35MB, dintre care 10MB sunt ocupati de texturile aflate in joc.



* **Audio**

Memoria dedicata fisierelor audio este de aproximativ 6.9MB, ridicandu-se pana la un total de 7.1MB in scena de joc, in timpul animatiilor. Acestea ocupa maxim 2% din totalul Audio CPU.

