Atestat de competențe profesionale la Informatică
Mai 2021

ATESTAT

Profesor coordonator: Mușunoiu Iuliana Monica Novetschi

Autor: Stoica Viorel

Atestat de competențe profesionale la Informatică
Mai 2021

Crearea unui joc în Unity

Profesor coordonator: Mușunoiu Iuliana Monica Novetschi

Autor: Stoica Viorel

Colegiul Național “Mihail Kogălniceanu”, Galați

Conținut

Atestat de competențe profesionale la Informatică
Mai 2021

Prin această lucrare de atestat am descris în detaliu procesul de creare al unui joc tridimensional folosind Unity și programând cu Visual Studio în limbajul C#. Produsul final conține 3 nivele ce au complexitate crescătoare, audio complet, meniuri ușor de înțeles, un generic de final după al treilea nivel și setări de bază. Nivelul sunetului, rezoluția și sincronizarea jocului cu ecranul pot fi schimbate foarte ușor din meniul de opțiuni.

Am ales să fac un joc deoarece impune lucrarea și în afara unui mediu cunoscut, interacționarea cu programe și sisteme noi, și eventual înțelegerea acestora la un nivel ce permite lucrarea eficientă cu acestea. Deoarece programatul presupune lucrarea cu multe software-uri diferite în multe ipostaze diferite, crearea unui joc era cel mai apropiat lucru de aceasta. Combinația de lucrat la o lume tridimensională și de scris cod ce se va îmbina cu această lume este o provocare unică, ce impune multă gândire pentru a găsi cea mai bună soluție la eventualele probleme găsite.

Având o pasiune pentru jocuri din tinerețe, mereu am vrut să dezvolt un joc. În liceu am învățat tehnicile de programare necesare pentru a putea realiza acest lucru cu succes, iar această lucrare de atestat s-a dovedit a fi oportunitatea perfectă pentru împlinirea unei dorințe.

Descrierea jocului

Jocul are un concept de bază simplu: trebuie să duci cubul până la capătul nivelului fără a lovi obstacole. Jucătorul trebuie doar să evite sau să sară peste obstacole pe măsură ce cubul înaintează. Provocarea jocului însă vine din felul în care este manipulată gravitația pentru a permite deplasarea pe pereți și tavane, făcând jocul mai diferit, mai interesant și, în cele mai multe situații, mai greu. Aceasta, în combinație cu caracterul simplist al graficii creează un joc scurt dar distractiv și cu o greutate progresivă.

Jucătorul poate controla cubul cu doar 3 taste, făcând conceptul mai accesibil și pentru persoane ce nu sunt obișnuite cu calculatoarele sau jocurile. Folosind săgețile sau tastele A și D se controlează mișcarea cubului în stânga și dreapta, iar tasta spațiu controlează săritura cubului. Cubul accelerează singur iar viteza acestuia nu poate fi modificată de jucător în mod direct, pentru a reduce din complexitatea controalelor și pentru a menține un anumit nivel de consistență de-a lungul nivelelor. Tastele sunt programate cu un sistem simplu dar robust ce nu permite pierderea informației de la taste, adică input loss. Mouse-ul se folosește doar în afara nivelelor, pentru a naviga meniurile.

Având în vedere puterea limitată de procesare a calculatoarelor disponibile, timpul limitat și complexitatea pe care ar impune-o un stil grafic realistic, am optat să fac jocul într-un stil “low poly”, ce implică folosirea unor culori și modele 3D simple în favoarea performanței si accesibilității.

Low Poly

Normal

Acest stil implică reducerea considerabilă a numărului de poligoane ale unui model 3D pentru a câștiga performanță și pentru a simplifica scena sau mediul de lucru, deoarece aceste modele nu necesită, în general, la fel de multă muncă pentru a le face să arate bine odată ce au fost introduse.

Unity Engine și Visual Studio

Unity este o platformă de dezvoltare cunoscută și robustă, lansată în 2005, ce pune accentul foarte mult pe compatibilitate, ușurința de a o folosi și gama largă de sisteme de operare suportate. Platforma suportă atât medii 3D cât și 2D, are un magazin integrat de modele, atât cu licențe cât și gratuite, și oferă multă flexibilitate în majoritatea aspectelor acestuia. Motivul principal pentru care am ales Unity este accesibilitatea oferită. Dezvoltatorii lui mențin o bază de date foarte bine structurată despre toate funcțiile engine-ului, de aceea multe din problemele întâmpinate în timpul dezvoltării jocului au putut fi rezolvate cu ajutorul ei.

Un alt motiv pentru care am ales Unity în locul altei platforme este faptul că acesta folosește C# ca limbaj principal de programare suportat. Deoarece C# este oarecum asemănător cu C++, m-am acomodat ușor cu acesta, fără sa fie nevoie de mult timp să îl înțeleg. C# este un limbaj de programare modern și simplu, cu o sintaxă familiară: acoladele înconjoară instrucțiunile funcțiilor, vectorii sunt declarați și folosiți cu paranteze pătrate, instrucțiunile se încheie cu punct și virgulă.

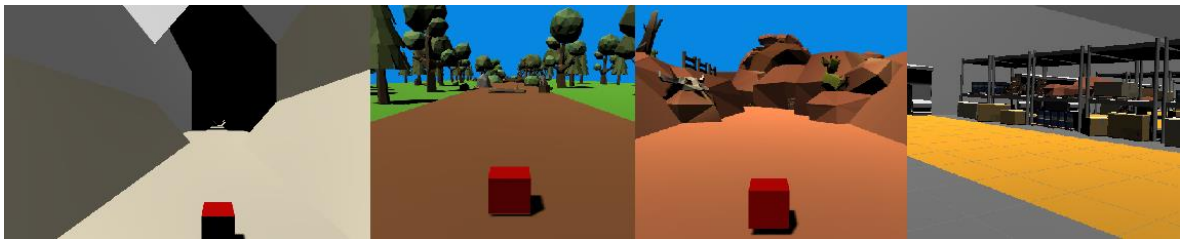
Pentru a scrie codul in C#, am folosit Visual Studio. Este un mediu de dezvoltare rapid, simplu, plăcut, ce nu obosește vederea cu fonturi greu de înțeles. Are suport integrat pentru Unity și C#, ceea ce îl face un mediu excelent. De asemenea, pune la dispoziție unelte foarte folositoare pentru formatare, rescriere rapidă și comentarii. Sugestiile de cod sunt bune la detectat posibile greșeli din timp, făcând programatul mai plăcut iar culorile diferențiază părțile de cod cu ușurință.

Deoarece au suport oficial reciproc, Unity permite transmiterea de variabile din editor direct către script foarte ușor și are un compilator integrat ce scoate în evidență erorile din codul scris. Visual Studio poate recunoaște funcțiile din Unity și le dă culoarea corespunzătoare, ajutând cu lizibilitatea generală a codului. Posibilitatea de a schimba o variabilă dintr-un script fără a modifica liniile de cod direct din editorul Unity este alt avantaj al acestor două programe.

Originile ideii

Ideea pentru acest joc a venit inițial de la un alt joc, Flappy Bird, ce are o premisă asemănătoare: trebuie să te ferești de obstacole controlând un caracter. Am luat conceptul acesta și l-am combinat cu un alt concept de la un alt joc numit Run. În acela, caracterul controlat trebuia să meargă pe podelele, pereții și tavanele nivelului pentru a ajunge la capăt. Deși primul era 2D iar al doilea 3D, am putut să intercalez toate părțile importante din fiecare și să le introduc în jocul meu.

Inițial, încercasem un design minimalist asemănător cu cel din Run, dar mediul era prea monoton și ar fi trebuit să mă bazez pe o dificultate sporită pentru a face jocul mai interesant. Am optat eventual să fac doar 3 nivele, dar acestea să fie mult mai bine făcute decât cel inițial. Am dat fiecăruia o tematică pentru a face diferența dintre ele mai sesizabilă și pentru a permite diferite tipuri de obstacole. Am ales o tematică de pădure, una de deșert și una industrială și am făcut obstacole corespunzătoare fiecăreia, făcând locațiile să arate puțin mai natural.

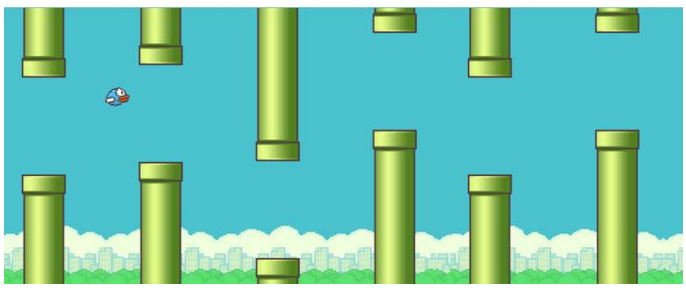


Prototip Pădure Deșert Industrial

Am ales cubul roșu drept caracter inițial pentru simplitate, deoarece era mai important să am un prototip funcțional mai întâi, dar am ales să îl păstrez pentru că iese în evidență față de restul obiectelor iar forma simplă reprezintă o constantă plăcută de-a lungul nivelelor. Culoarea roșie nu este folosită aproape deloc în restul jocului, deci nu se intersectează cu restul nivelelor.

Flappy bird este un joc pentru telefoane ce este cunoscut pentru nivelul ridicat de dificultate. În contrast cu acest joc, Flappy bird nu are un sfârșit, ci are un leaderboard cu scorurile obținute de jucător. Jocul a fost lansat pe 24 mai și are grafică 2D. Apăsând ecranul telefonului pasărea este lansată în sus, iar sincronizarea săriturii permite depășirea obstacolelor. Run, pe de altă parte, nu folosește obstacole propriu-zise pentru a adăuga dificultate, bazându-se pe o combinație de reflexe rapide și de gândire logică. Are grafică tridimensională, și deși este mult mai rudimentară, se potrivește foarte bine cu atmosfera.

Flappy Bird



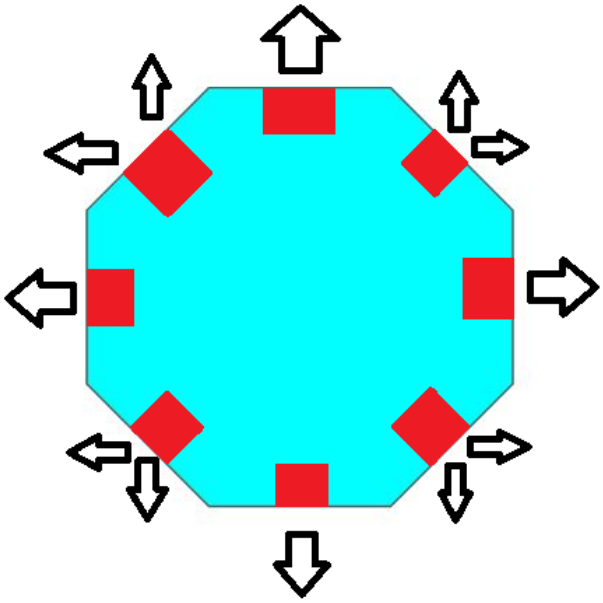
Run

Explicarea codului

Codul jocului este compus din mai multe scripturi ce sunt atașate de obiecte din nivele sau meniuri. Există un script principal (“Miscare.cs”) pentru toate cele 3 nivele ce conține majoritatea interacțiunilor fizice și animația camerei. Incluzând comentariile, acest script are în jur de 330 de linii de cod. Celelalte scrpituri pe care le folosesc sunt: “Meniuprincipal.cs”, ce include liniile de cod necesare pentru funcționarea butoanelor din meniul de nivele și a celui de ieșire; Scriptul “ScriptOptiuni.cs” este legat de funcționalitatea butoanelor din meniul de opțiuni și conține funcțiile pentru Vertical sync, Fullscreen, rezoluții și nivelul sunetului. Scripturile “endlevelcollider.cs”, “rotire.cs”, “usa.cs” și “rotirex.cs” fac lucruri mai puțin importante, cum ar fi rotații decorative, mișcări și returnări la meniu.

Scriptul Principal, sau “Miscare.cs”, are în total 18 variabile și 5 obiecte inițializate prin referință. Cele 4 funcții principale din el sunt Start, OnCollisionEnter, Update și FixedUpdate. Funcția Start se execută o singură dată, prima oară când este activat scriptul. Start conține inițializările pentru vectorul de direcție al gravitației. OnCollisionEnter se execută de fiecare dată când cubul intră în contact cu orice alt corp. Această funcție are rolul de a verifica pe ce suprafață se află corpul, pentru a-i putea aplica forțele corespunzător. Pentru a putea înțelege mai ușor ce face funcția, e necesară o explicație a felului în care funcționează jocul.

Am stabilit de la început că vreau ca mișcarea cubului să fie posibilă pe suprafețe la 8 unghiuri diferite prestabilite, cu diferențe de 45 de grade între ele. Fiecare suprafață este numerotată în funcție de unghiul ei, de la “Plan1” la “Plan8”. În imagine cubul roșu este reprezentat pe toate cele 8 planuri implementate în joc. Cubul poate merge doar pe interiorul acestora, dar ele pot fi de orice lungime sau lățime dorită. De asemenea, suprafețele nu trebuie să formeze neapărat un octagon ca în imagine, ele pot fi la distanțe variate unele de celelalte, lucru pe care l-am folosit spre avantajul meu de câteva ori în crearea nivelelor.



Asupra cubului este aplicata forța de greutate în funcție de suprafața pe care acesta stă. Dacă el stă pe o suprafață de sus, jos, din stânga sau din dreapta, greutatea este aplicată cu o singură forță pe direcția săgeților din imagine. Dacă stă pe una din suprafețele la unghi de 45 de grade, atunci greutatea va fi compusă din 2 forțe pe direcția săgeților. Forțele ce vor compune greutatea vor avea valoarea de 0.71*greutate, pentru ca rezultatul lor să fie egal cu cel al greutății normale. Această soluție este folosită datorită limitărilor impuse de Unity.

Funcția **OnCollisionEnter** verifică numele obiectului cu care s-a intersectat cubul și va reține forța de pe axele corespunzătoare pentru ca alte funcții să imite gravitația pe fiecare plan în mod consistent. Dacă însă ea detectează că a fost o coliziune cu un obstacol, adică orice nu e numit "Plan" cu un număr la capăt, atunci funcția va reseta corpul la poziția inițială, resetând totodată și rotația, planul stocat în variabile și rotația rămasă.

```
24      void OnCollisionEnter(Collision col)
25      {
26          oksup = true;
27          switch (col.gameObject.name)
28          {
29              case "Plan1":
30                  {
31                      Debug.Log("1");
32                      suprafata = 1;
33                      fy = -FortaVertical;
34                      fx = 0;
35                      break;
36                  }
37              case "Plan2":
38                  ...
45              case "Plan3":
46                  ...
53              case "Plan4":
54                  ...
61              case "Plan5":
62                  ...
69              case "Plan6":
70                  ...
77              case "Plan7":
78                  ...
85              case "Plan8":
```

Oksup este folosit pentru a reține dacă a atins o suprafață și este folosit mai târziu de o altă funcție. Instrucțiunea Switch verifică simultan dacă numele suprafeței cu care a intrat corpul în contact este unul din cele 8 planuri ce determină direcția greutății. **Fy** și **fx** setează mărimea forței de greutate pe axele de coordonate respective, iar variabila **suprafata** retine care din planuri a fost atins ultima dată de cub. Fiecare caz al funcției este o variație puțin diferită, unde diferă doar variabilele **fx** și **fy**. Deoarece scriptul este atașat de cubul roșu în editorul Unity, **gameObject.name** va returna mereu numele planelor sau obstacolelor cu care interacționează.

```
101 default:
102     {
103         Debug.Log("Obstacol");
104         suprafata = 1;
105         suma = 0;
106         fy = -FortaVertical;
107         fx = 0;
108         supraftrec = suprafata;
109         playertransform.rotation = Quaternion.Euler(playertransform.rotation.eulerAngles.x, playertransform.rotatio
110         Cameratransform.rotation = Quaternion.Euler(Cameratransform.rotation.eulerAngles.x, Cameratransform.rotatio
111         resetpoz.x = 1f;
112         resetpoz.y = 1f;
113         resetpoz.z = 15f;
114         playertransform.position = resetpoz;
115         break;
116     }
```

Cazul **default** este folosit pentru când nu este o coliziune cu un plan, aceasta resetând variabilele **fx**, **fy**, **suprafata**, **supraftrec** și **suma**, folosite de alte funcții. De asemenea, readuce corpul și camera la poziția și rotația inițială.

Funcția **Update** are rolul de a transmite inputul de la jucător către restul programului. Deoarece Update este executat mai des decât celelalte funcții, este cea mai potrivită pentru a lua input și a-l stoca în variabile pentru a fi folosit de către alte funcții. Aceasta verifică starea tastelor W, S, Spațiu, stânga, dreapta și escape și le stochează starea în variabile bool pentru a fi folosite in FixedUpdate.

```
120     void Update()
121     {
122         if (Input.GetKey("a")||Input.GetKey("left"))
123             oka = true;
124         if (Input.GetKey("d")||Input.GetKey("right"))
125             okd = true;
126         if (Input.GetKeyDown(KeyCode.Space))
127             okspace = true;
128         if (Input.GetKey(KeyCode.Escape))
129             okescape = true;
130     }
```

Este necesară stocarea stării tastelor în variabile deoarece fără acestea există o șansă semnificativă de a nu se înregistra unele apăsări ale tastelor, în special cele de scurtă durată. Variabilele **oka**, **okd**, **okspace** și **okescape** stochează valorile pentru tastele A, D, stânga, dreapta, spațiu și escape. **Oka** stochează valoarea pentru A și stânga simultan deoarece ele au același scop. **Okd** stochează valoarea pentru D și dreapta din același motiv.

Funcția **FixedUpdate** este cea mai mare și mai complexă din toate, deoarece ea îmbină restul funcțiilor. Primul lucru pe care îl face este să verifice dacă s-a schimbat suprafața de la ultima verificare și, dacă da, să crească un contor ce va determina rotirea corpului.

```
void FixedUpdate()
{
    //invart corpul si camera
    if(suprafata!=supraftrec)
    {
        if((supraftrec > suprafata&&(supraftrec!=8||suprafata!=1))||(supraftrec==1&&suprafata==8))
        {
            modif = -5f;
            suma = suma - 45f;
        }
        else if(supraftrec < suprafata||supraftrec==8&&suprafata==1)
        {
            modif = 5f;
            suma = suma + 45f;
        }
    }
```

Se testează dacă s-a schimbat suprafața de la ultima verificare comparând **suprafata** și **supraftrec**. Dacă sunt diferite va compara valorile celor două variabile pentru a determina în jurul cărei axe trebuie să se execute rotația corpului. Variabila **modif** reține viteza și totodată direcția în care va fi rotit corpul. **Suma** este un contor ce arată câte grade trebuie rotit corpul în total, și e folosit in alte funcții.

Al doilea pas este să învârtă corpul, iar camera opus corpului. **Playertransform** și **cameratransform** sunt modificate pe axa z cu +**modif**, respectiv +2*__modif__*(-1). Variabila **suma** este decrementată cu valoarea din **modif** pentru a reflecta rotația făcută. Acest sistem ce ține cont de cât trebuie să parcurgă în mod total este făcut pentru a permite corpului să schimbe rapid suprafețe fără vreo problemă. În cazul în care se schimbă o suprafață înainte de a se termina rotația ce se face pe parcursul a mai multe cadre, acest sistem asigură o rotație completă ce își poate schimba direcția în timpul rotirii.

```
if (suma!=0)
{

    playertransform.rotation = Quaternion.Euler(playertransform.rotation.eulerAngles.x, play
    Cameratransform.rotation = Quaternion.Euler(Cameratransform.rotation.eulerAngles.x, Came
    suma = suma - modif;
}
directieg = new Vector3(fx, fy, fz);
```

Următorul switch va aplica forțele laterale și cea de săritură în funcție de suprafață și de variabilele de la funcția Update. Conține 8 cazuri, ca **OnCollisionEnter,** pentru fiecare plan. Se verifică fiecare tastă apăsată și i se resetează valoarea **oka**, **okd**, **okspace** sau **oksup** în funcție de caz. Primul if adaugă forță laterală pentru a muta cubul în stânga cu valoarea lui **-FortaLateral.** Al doilea if aplică forță cu valoarea +**FortaLateral,** iar al treilea adaugă forță vertical pentru săritură în funcție de **FortaSaritura** testând dacă a atins solul de la ultima săritură. Sistemul acesta are o problemă mică, deoarece testează doar atingerea solului. În cazul în care cubul ar cădea de pe o suprafață, îi permite să sară și din aer o singură dată. Această problemă este totuși rezolvată prin level design , neavând nevoie de un script mai complex sau încet pentru a rezolva problema.

```
156         switch (suprafata)
157         {
158             case 1:
159                 {
160                     if(oka)
161                     {
162                         Player.AddForce(-FortaLateral, 0, 0);
163                         oka = false;
164                     }
165                     if(okd)
166                     {
167                         Player.AddForce(FortaLateral, 0, 0);
168                         okd = false;
169                     }
170                     if (okspace && oksup)
171                     {
172                         Player.AddForce(0, FortaSaritura, 0);
173                         okspace = false;
174                         oksup = false;
175                     }
176                     break;
177                 }
178             case 2:
```

Ultimul lucru pe care îl face este să aplice forța de gravitație pe vectorul dat de funcția OnCollisionDetection, să aplice forța de înaintare și, dacă a fost apăsat escape, să se întoarcă la meniu. Variabila **supraftrec** este folosită pentru a reține suprafața din cadrul trecut și e folosită de FixedUpdate, iar aici este reinițializată. **Okspace** este de asemenea resetat pentru stabilitatea funcției de sărit. **Directieg** este vectorul directiei pe care trebuie aplicată greutatea și este calculat mai sus, aici fiind aplicată. **FortaFata** reprezintă valoarea forței cu care cubul este propulsat în față.

```
319             supraftrec = suprafata;
320             //Miscam playerul
321             Player.AddForce(0, 0, FortaFata);
322             //Gravitatia
323             Player.AddForce(directieg);
324             //Saritura
325             okspace= false;
326             if(okescape==true)
327             {
328                 okescape = false;
329                 SceneManager.LoadScene(0);
330             }
331         }
332     }
333
```

Următorul script ca mărime și complexitate este **ScriptOptiuni.cs**, scopul codului este să asigure funcționarea listei de rezoluții, verificarea rezoluțiilor disponibile în funcție de monitor, setarea volumului, rezoluției, fullscreen și vsync. În funcția Start se detectează rezoluțiile și se stochează într-o listă pentru a fi folosite de meniu.

```
for(int i=0;i<resolutions.Length;i++)
{
    string option = resolutions[i].width + "x" + resolutions[i].height;
    options.Add(option);
    if(resolutions[i].width==Screen.currentResolution.width&&resolutions[i]
    {
        currentResolutionIndex = i;
    }
}
```

SetResolution este aplicată în editor listei de rezoluții și setează rezoluția în funcție de numele opțiunii selectate. **SetVolume** e folosit pentru a selecta volumul global între scene. **SetFullscreen** schimbă fereastra aplicației între windowed și fullscreen. **SetVsync** schimbă rata de reîmprospătare a aplicației pentru a fi aceeași cu cea a ecranului, eliminând desincronizarea dintre zonele ecranului.

QualitySettings.vSyncCount este o funcție specială integrată în Unity ce determină cât de des să se sincronizeze imaginea cu ecranul calculatorului pentru a elimina impresia de imagine tăiată. Valoarea 1 înseamnă că va sincroniza jocul la rata de reîmprospătare nativă a ecranului, în cele mai multe cazuri aceasta fiind de 60Hz. Valoarea 0 va dezactiva sincronizarea, lăsând placa video să lucreze la capacitate maximă. Am introdus această opțiune deoarece pe un număr limitat de calculatoare și monitoare desincronizarea este foarte semnificativă și strică experiența jocului.

```
public void SetResolution(int resolutionIndex)
{
    Resolution resolution = resolutions[resolutionIndex];
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
}
public void SetVolume(float volume)
{
    Debug.Log(volume);
    audioMixer.SetFloat("volume", volume);
}
public void SetFullscreen(bool isFullscreen)
{
    Screen.fullScreen = isFullscreen;
}
public void SetVsync(bool isVsyncOn)
{
    if(isVsyncOn)
    QualitySettings.vSyncCount = 1;
    else
    QualitySettings.vSyncCount = 0;
}
```

Scriptul endlevelcollider.cs este alcătuit dintr-o singură funcție OnTriggerExit ce încarcă meniul principal atunci când este activată. Scena 0 reprezintă meniul principal, iar scenele 1, 2 și 3 sunt nivelele jocului.

```
void OnTriggerExit(Collider other)
{
    Debug.Log("Exit");
    SceneManager.LoadScene(0);
}
```

Scripturile rotire.cs și rotirex.cs sunt folosite pentru rotații în primul și al treilea nivel, singura diferență dintre ele fiind axa în jurul căreia se învârt corpurile, deoarece rotația în jurul axei x este făcută puțin diferit. Variabilele **rotirex**, **rotirey** și **rotirez** determină viteza rotirii pe cele 3 axe.

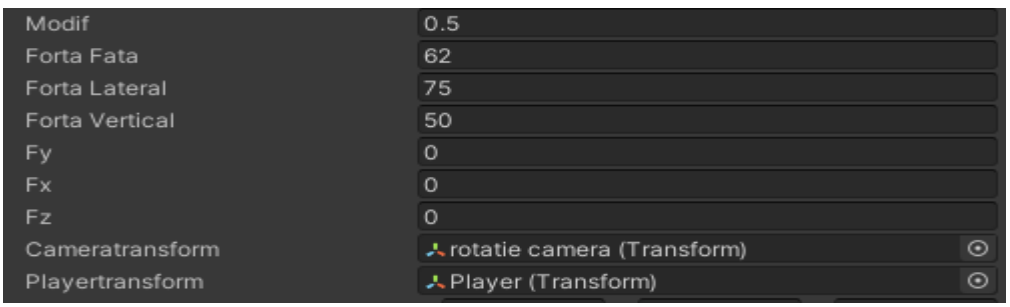
```
public Transform corp;
public float rotirex,rotirey,rotirez;
void FixedUpdate()
{
    corp.rotation = Quaternion.Euler(corp.rotation.eulerAngles.x + rotirex, corp.rotation.eulerAngles
}
```

Scriptul Usa.cs este folosit doar în al treilea nivel pentru a deschide ușile dintre camere. Funcția OnTriggerEnter se activează în proximitatea ușilor deoarece am modificat raza de detecție a acestora în editorul Unity. Aceasta schimbă o variabilă de tip bool iar FixedUpdate schimbă progresiv poziția corpului pe verticală pe parcursul a mai multor cadre.

```
public Transform usa;
public int ok=0;
public float viteza;
void OnTriggerEnter()
{
    ok = 1;
    Debug.Log("Usa");
}
void FixedUpdate()
{
    if(ok==1)
    {
        usa.position = usa.position + new Vector3(0f, viteza, 0f);
    }
}
```

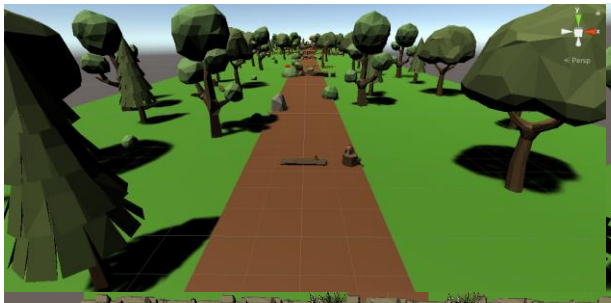
Unele dintre variabilele scripturilor nu sunt inițializate în interiorul codului, acestea sunt inițializate prin intermediul editorului Unity pentru a oferi o flexibilitate mai mare. De exemplu, forța cu care corpul este împins în față variază de la nivel la nivel pentru a ajusta dificultatea într-un mod subtil dar eficient. Acest lucru a ajutat mult și cu testarea jocului în general, deoarece permite izolarea unor probleme pentru a le rezolva mai ușor.

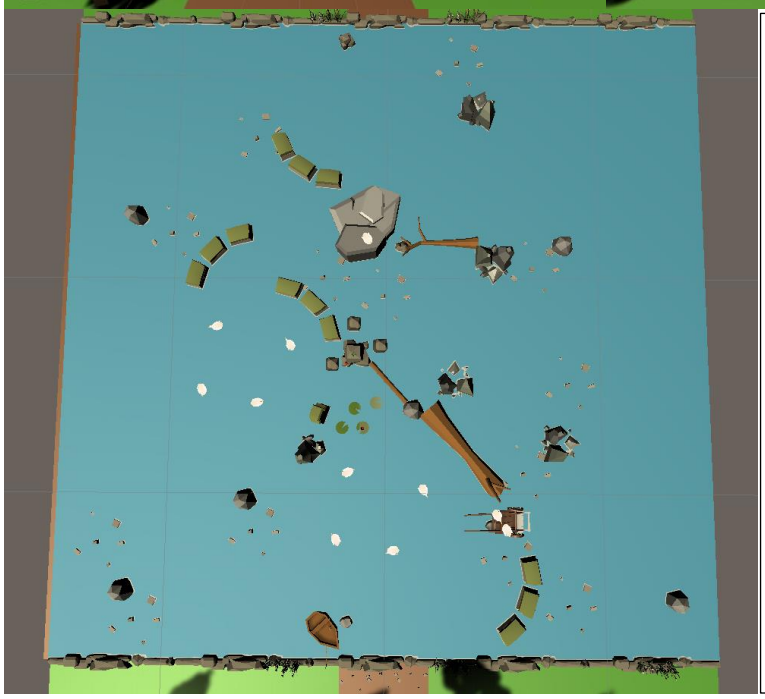
Pentru toate funcțiile ce afectează un corp sau funcționează pe baza datelor de la un corp, acestea au nevoie de o referință în editorul Unity la componenta corpului respectiv. De cele mai multe ori, referința se face la componenta “Transform” a unui corp, deoarece aceasta conține poziția și rotația corpului, acestea fiind cele mai folosite componente. O altă componentă ce a primit destul de multe referințe este rigidbody, ce este folosit pentru crearea de coliziuni între corpuri sau detectarea lor.



Nivelele jocului

Primul nivel al jocului este nivelul din pădure, unde cubul merge pe un drum de pământ printre copaci, pietre și lemne căzute. Acest nivel este și singurul ce nu folosește mecanica planurilor ce schimbă "gravitația", deoarece am vrut să fie un nivel în care jucătorul să se poată obișnui cu mișcarea cubului, viteza lui, evitatul obstacolelor și obiectivul său. Nivelul începe cu niște obstacole simple, vizibile și ușor de evitat. Zona îngrădită este primul obstacol unde se pot alege mai multe rute. Un jucător poate sări peste gard, ceea ce este mai ușor și mai evident, dar îi va face următoarea săritură mai grea. A doua opțiune, mai puțin evidentă, este să ocolească gardul prin spațiul dintre cele doua bucăți ce îl alcătuiesc. Este mai greu de realizat mișcarea pentru a trece prin spațiul acesta, dar face mult mai ușoară săritura ce urmează.





Ultima parte importantă a nivelului este și cea mai grea, partea lacului. Sunt în total trei feluri de a trece de ea: printr-o gaură din gardul din mijloc evitând rațele dinaintea lui, printre rațele din stânga pietrelor sau făcând un slalom strâns între lemnele și pietrele din dreapta. Cea mai scurtă cale este cea din mijloc, dar este destul de greu să nu lovești nimic. A doua cea mai scurtă este cea din dreapta, iar cea din stânga este cea mai lungă, cu o dificultate comparabilă celei din dreapta.

Rațele din lac se rotesc în jurul unui punct fix cu ajutorul funcției **rotire.cs**, iar cele două grupuri de rațe se rotesc în sensuri opuse unul față de celălalt. Pietrele din lac au rolul de a reduce vizibilitatea și de a delimita cele trei trasee de parcurgere a nivelului.

Odată ce a trecut de lac, jucătorul ajunge la portalul de final, unde va fi readus la meniul principal al jocului. Această întoarcere este realizată cu scriptul **endlevelcollider.cs**.

Al doilea nivel este nivelul din deșert, unde cubul merge printr-un teren uscat, din piatră portocalie, traversează un apeduct și un pod de lemn și explorează la final o mină părăsită. În acesta apar și planurile ce schimbă gravitația pentru prima dată, adăugând complexitate. Începutul are o piatră mare ce blochează drumul normal, forțând jucătorul să meargă pe un plan din stânga, astfel introducând conceptul de schimbare a gravitației. Cubul apoi va trece pe sub un apeduct și va avea primul obstacol principal după acesta.

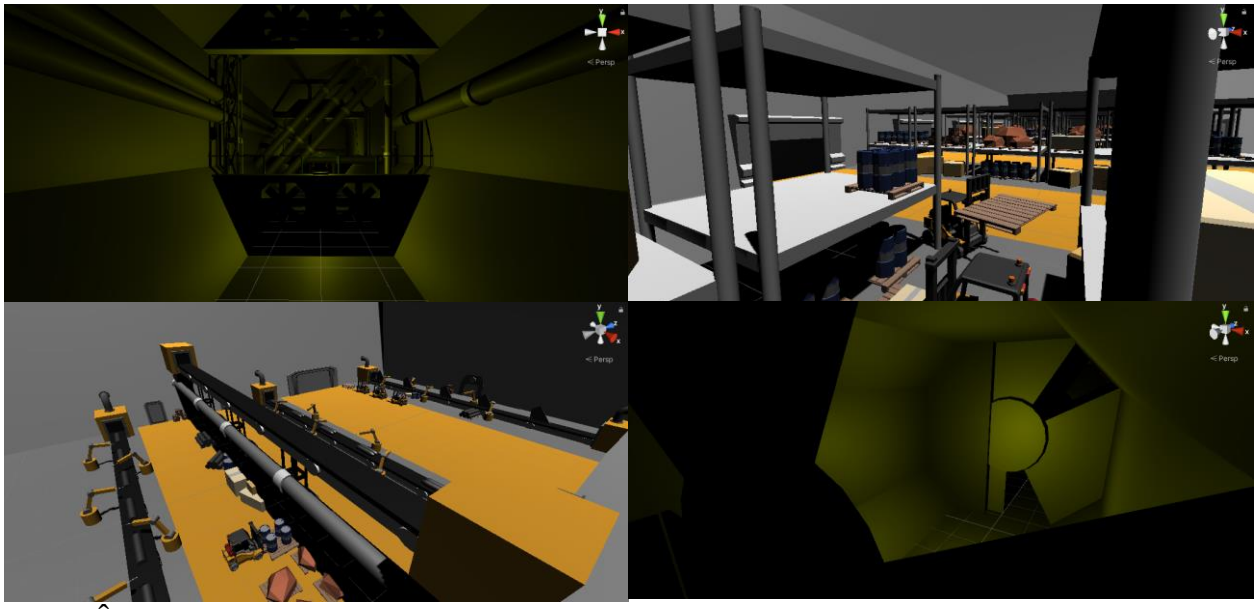
Din reflex, jucătorul va încerca să continue pe sub pod, dar calea este blocată. În schimb, trebuie să urce pe cupola de piatră, sărind în vârful ei pe o a doua cupolă de piatră ce îi permite să se reîntoarcă la nivelul solului. Cupola implică o rotație completă de 360 de grade, folosind toate planurile de gravitație implementate în sistemul jocului.

Următorul obstacol este o scară ce trebuie urcată cu atenție, altfel jucătorul nu va avea timp să o sară complet, lovind un obstacol. Aceasta va urca jucătorul la nivelul cactușilor, putând astfel observa deșertul și tabăra de dinaintea minei. Intrând în mină, cubul trebuie să se ferească de pietrele tavanului căzut și de materialele lăsate în urmă pentru a ajunge la ușa de metal de la finalul nivelului.

Acesta este primul nivel ce folosește și surse secundare de lumină pe lângă cea globală ce simulează soarele, pentru a ilumina mina. Am folosit un număr limitat pentru a nu afecta considerabil performanța legată de placa video, dar am introdus destule pentru a păstra aspectul și atmosfera nivelului. Cactușii și copacii sunt folosiți doar în zonele unde jucătorul i-ar putea vedea, nu pe toată lungimea planului, pentru a nu procesa obiecte nesemnificante.

Dificultatea nivelului este reflectată în introducerea obstacolelor mai complexe, reducerea distanței dintre ele și nevoia de a gândi diferit pentru a avansa în partea inițială a nivelului. Muzica este de asemenea unul din factorii importanți ai atmosferei, potrivindu-se cu stilul vizual.

Al treilea nivel este nivelul industrial, cu un caracter mult mai închis, fiind singurul nivel complet înconjurat de pereți. Nivelul de asemenea se folosește extensiv de obstacole formate din planurile de gravitație și include cel mai mare număr de obiecte din cele trei. Ca nivel de dificultate, este cel mai greu și necesită mai multă concentrare pentru a-l putea termina. Jucătorul începe într-un tub de ventilație unde trebuie să evite un amalgam de țevi și ventilatoare pentru a ajunge la o ușă în capătul tunelului. Aceasta se deschide și cubul intră într-o zonă de stocare cu motostivuitoare.



În zona de stocare jucătorul trebuie să sară pe rafturi și să meargă pe sub rafturi pentru a ajunge la a doua ușă ce desparte zonele. Unul din obstacolele grele ale nivelului este aici, deoarece după primul raft calea de continuare este acoperită de o umbră, făcând traversarea mult mai grea la început.

După a doua ușă se află o zonă de asamblare, cu benzi rulante statice, brațe robotice și materiale de construcție dezorganizate. Este un loc mai ușor de traversat decât celelalte în mod intenționat, pentru a oferi o mică pauză jucătorului înainte de următoarea zonă.

Zona finală este alcătuită din trei ventilatoare de mărimea întregului tunel, cu unul din ele fiind pe jumătate blocat de un perete, forțând jucătorul să folosească schimbarea gravitației pentru a ocoli obstacolul în cele mai multe cazuri. Ventilatoarele se rotesc la viteze și în direcții diferite făcând această ultimă zonă un test destul de greu de trecut. În general, acest nivel este de o dificultate considerabilă dacă este comparat cu celelalte două.

La sfârșitul acestei zone este o ultimă ușă, ce duce în exteriorul locației, apoi începe rularea genericului, unde sunt menționate toate persoanele ce au ajutat cu testarea jocului, găsirea de probleme în designul nivelelor. Au fost un ajutor bun pentru a determina dacă o zonă este prea grea, prea ușoară, prea monotonă sau prea încărcată, astfel contribuind la produsul final într-un mod indirect dar important și apreciat.

Optimizarea

În decursul scrierii codului am adus optimizări atât scripturilor cât și obiectelor din nivele pentru a reduce numărul de operații efectuate de procesor și de placă video. Cea mai notabilă diferență este reducerea funcțiilor de tip switch din scriptul miscare.cs de la șase la doar două. Primele versiuni ale codului verificau suprafața pe care se află corpul de fiecare dată când se detecta input de la tastatură. Combinarea tuturor funcțiilor în una mai mare s-a dovedit a fi mai eficient și ca timp, și ca spațiu. În plus, era mult mai ușor să fac schimbări mari la cod după ce am centralizat părțile importante din cod.

Altă optimizare adusă a fost să reduc numărul de variabile necesare. După ștergerea liniilor de cod nefolosite și revizuirea părților ineficiente, am putut să reduc la jumătate numărul de variabile folosite.

În interiorul editorului Unity, am dezactivat componentele de coliziune pe obiectele ce erau departe de jucător sau care erau prea mici pentru a fi relevante. Obiectele ce sunt mai departe de o anumită distanță de către cameră nu mai sunt vizibile, pentru a economisi resurse. O altă proprietate pe care am modificat-o este distanța la care camera afișează obiecte. În primul și al doilea nivel este setată la 900, iar în al treilea la 500. Deși în primul și al doilea nivel nu se poate folosi prea mult din cauza naturii deschise a lor, al treilea nivel a primit un bonus major în performanță de la această setare, deoarece nivelul are o natură secționată. În comparație cu versiunea lui inițială, această setare a crescut performanța cu aproximativ 200%, în special la partea inițială a nivelului, unde performanța are cel mai mult de suferit.

Specificațiile recomandate sunt un procesor i7 6700HQ la o frecvență de 2.4Ghz cu o placă video integrată Intel HD Graphics 530, ambele fiind piese lansate în 2015, cu opțiunea de vsync pornită pentru a limita cadrele pe secundă și a monitoriza mai ușor performanța. Primul și al doilea nivel nu au solicitat foarte mult componentele, iar al treilea nivel a adus placa video la 90% utilizare. Astfel, am reușit să rulez jocul doar cu o placă video integrată din punct de vedere grafic, atingându-mi scopul inițial. În cazul în care placa video totuși nu face față, scăderea rezoluției din interiorul meniului de opțiuni poate face o diferență foarte mare în numărul de cadre pe secundă.

Concluzie

Procesul de creare al unui joc în Unity este unul complex, ce necesită mult timp și răbdare, dar cu perseverență se poate atinge un rezultat foarte bun. Odată familiarizat cu interfața Unity și cu felul în care se utilizează, munca poate fi concentrată mai mult pe adăugat conținut și mai puțin pe rezolvat probleme. Scenele se compun ușor, având la dispoziție modele 3d prin magazinul de modele. Fiecare obiect are o multitudine de proprietăți ce pot fi schimbate pe placul programatorului. Optimizarea jocului se poate face manual, sau automat, scăzând timpul de creare al jocului în funcție de nevoi. În concluzie, orice persoană cu dedicație și cunoștințe de bază de programare poate crea un joc simplu cu ajutorul Unity.

Bibliografie

- https://en.wikipedia.org/wiki/Unity_(game_engine)
- https://ro.wikipedia.org/wiki/Microsoft_Visual_Studio
- https://stackoverflow.com
- https://docs.unity3d.com/Manual/index.html
- https://docs.microsoft.com/ro-ro/visualstudio/windows/?view=vs-2019