„Taxi Simulator” folosind Unity

Frînculeasa Alexandru

1.Tehnologii folosite

3.1)HoudiniFx

Houdini este o aplicație de animație 3D și efecte speciale dezvoltată de Side Effects Software, o companie veche de douăzeci și cinci de ani din Toronto. Houdini a fost conceput pentru artiștii care lucrează în animație 3D și VFX pentru film, TV, jocuri video și realitate virtuală. Houdini reunește aceste lumi într-o singură platformă puternică.Fostul produs emblematic al Side Effects a fost PRISMS, o suită de instrumente grafice 3D care a servit ca bază pentru dezvoltarea Houdini. Spre deosebire de alte software-uri de animație 3D, Houdini folosește un flux de lucru procedural bazat pe noduri, care facilitează explorarea iterațiilor pe măsură ce îți perfecționezi munca. Programe precum Maya sau Blender stochează modificări într-un istoric al utilizatorilor, ceea ce face dificilă revenirea la o versiune anterioară a lucrării tale.Abordarea unică bazată pe noduri a lui Houdini permite mai multe iterații, astfel încât să poată face ușor modificări și să dezvolte animațiile și efectele. În timp ce Houdini este utilizat în principal pentru medii dinamice și efecte de particule, include un set complet de instrumente pentru artiștii care doresc să-l folosească pentru alte domenii, cum ar fi modelarea, animarea sau redarea conținutului.

Houdini este cel mai bine cunoscut pentru instrumentele sale avansate de simulare dinamică care permit crearea de efecte vizuale extrem de realiste. Houdini oferă un flux de lucru eficient conceput atât pentru studiourile mici, cât și pentru artiștii individuali. Noile optimizări ale software-ului permit artiștilor să obțină efecte de ultimă generație pe hardware mai puțin avansat.Ceea ce îl separă cu adevărat Houdini de alte programe de animație 3D este natura sa procedurală. Activele din Houdini sunt în general create prin conectarea unei serii de noduri. Avantajul acestui flux de lucru este că permite artiștilor să creeze obiecte detaliate într-un număr relativ scurt de pași, spre deosebire de alte programe. În timp ce Houdini este cunoscut pentru efectele speciale, acesta oferă toate instrumentele pe care v-ați aștepta să le găsiți într-un program 3D important. Houdini oferă modelare geometrică standard și animație prin cadre cheie. Programul este livrat cu puternicul motor de redare Mantra, dar acceptă și motoare de redare terță parte, cum ar fi Renderman.Houdini aduce o abordare procedurală a manipulării personajelor. Personajele pot fi împachetate într-un singur nod activ pentru a fi utilizate de o echipă de animație. Houdini are un sistem de fulgere bazat pe noduri, care oferă un mediu de

lucru flexibil pentru construirea umbrelor și crearea de efecte CG. Dispune de un sistem volumetric puternic pentru crearea simulărilor de fum și foc, precum și un compozitor pentru efecte de imagine stratificate. Houdini are un mediu deschis și oferă scripturi printr-o varietate de API-uri, dar Python este limba preferată pentru majoritatea pachetelor. Scriptarea internă permite automatizarea anumitor sarcini, precum și crearea de instrumente și pluginuri personalizate. Este un program foarte versatil! Natura procedurală a lui Houdini permite o dezvoltare neliniară, atât puternică, cât și flexibilă. Aspectele procedurale ale Houdini pot fi valorificate pentru a accelera producția și pur și simplu fluxul de lucru 3D.

Comportamentul procedural al Houdini oferă o evoluție neliniară care permite un flux de lucru puternic și flexibil. Această natură procedurală a software-ului ajută la accelerarea producției și crește fluxul de lucru 3D.

- Animație - Animația din Houdini acceptă animația cadrelor cheie și manipularea canalelor brute.
- Modelare - Modelarea în Houdini se bazează pe toate geometriile standard. Acestea sunt NURBS, Metaballs, Polygons și Bezier Curves.
- Iluminare - Se bazează pe un shader bazat pe noduri.
- Dinamică - Dinamica din Houdini constă în dinamica fluidelor, simularea pânzei, simularea mulțimii, dinamica firelor și dinamica corpului rigid.
- Redare - Folosește redarea sa puternică numită Mantra. Alte rendere utilizate sunt Octane, V-ray, Redshift și Renderman
- Compoziție - Se bazează pe imagini stratificate
- Dezvoltare plugin - Creați și dezvoltați biblioteci în funcție de nevoile utilizatorului.
- Scripting - O varietate de API-uri sunt utilizate în Houdini. Python este cel mai folosit limbaj.

Natura procedurală a lui Houdini se regăsește în operatorii săi. Activele digitale sunt în general construite prin conectarea secvențelor de operatori (sau OP-uri). Acest proceduralism are mai multe avantaje: permite utilizatorilor să construiască obiecte geometrice sau organice foarte detaliate în relativ puțini pași în comparație cu alte pachete; permite și încurajează dezvoltarea neliniară; iar noii operatori pot fi creați în termeni de operatori existenți, o alternativă flexibilă la scriptarea neprocedurală pe care se bazează adesea în alte pachete pentru personalizare. Houdini

folosește această paradigmă procedurală pe tot parcursul: pentru texturi, umbrere, particule, „date de canal” (date utilizate pentru a conduce animația), redare și compoziție.

Structura bazată pe operator a lui Houdini este împărțită în mai multe grupuri principale:

- OBJ - noduri care transmit informații de transformare (în mod tradițional acestea conțin SOP-uri).
- SOP - Operatori de suprafață - pentru modelarea procedurală.
- POP - Operatori de particule - utilizate pentru manipularea sistemelor de particule.
- CHOPs - Operatori de canale - pentru animație procedurală și manipulare audio.
- COP - Operatori de compoziție - folosit pentru a efectua compoziție pe filme.
- DOP - Operatori dinamici - pentru simulări dinamice pentru fluide, pânză, interacțiune rigidă a corpului etc.
- SHOPs - Shading Operator - pentru reprezentarea a zece sau mai multe tipuri diferite de umbrire pentru mai multe tipuri de redare diferite.
- ROP - operatori de redare - pentru construirea rețelelor pentru a reprezenta diferite treceri de redare și dependențe de redare.
- VOPs - Operatori VEX - pentru construirea nodurilor de oricare dintre tipurile de mai sus utilizând o arhitectură SIMD extrem de optimizată.
- TOP-uri - Operatori de sarcini [6]
- LOP - Operatori de iluminat - pentru generarea de USD care descrie personaje, recuzită, iluminare și redare.

Operatorii sunt conectați împreună în rețele. Fluxurile de date, manipulate de fiecare operator la rândul lor. Aceste date ar putea reprezenta geometrie 3D, imagini bitmap, particule, dinamică, algoritmi shader, animație, audio sau o combinație a acestora. Această arhitectură a grafică nodului este similară cu cea utilizată în compozitoarele bazate pe noduri, cum ar fi Shake sau Nuke.Rețelele complexe pot fi grupate într-un singur nod “meta-operator” care se comportă ca o definiție a clasei și poate fi instanțiată în alte rețele. În acest fel, utilizatorii își pot crea propriile instrumente sofisticate fără a fi nevoie de programare. În acest fel, Houdini poate fi privit ca un set de instrumente de programare vizuală extrem de interactiv, care face programarea mai accesibilă artiștilor.

Setul de instrumente al lui Houdini este în mare parte implementat ca operatori. Acest lucru a condus la o curbă de învățare superioară față de alte instrumente similare. Un lucru este să știi ce

fac toate nodurile - dar cheia succesului cu Houdini este înțelegerea modului de lucru și reprezentarea unui rezultat dorit ca o rețea de noduri. Utilizatorii de succes sunt, în general, familiarizați cu un repertoriu larg de rețele (algoritmi) care obțin rezultate creative standard. Cheltuielile generale implicate în achiziționarea acestui repertoriu de algoritmi sunt compensate de flexibilitatea artistică și algoritmică oferită de accesul la blocuri de construcție de nivel inferior cu care se configurează rutinele de creare a elementelor de fotografiere. În producțiile mari, dezvoltarea unei rețele procedurale pentru rezolvarea unei provocări specifice de creare a elementelor face ca automatizarea să fie banală. Multe studiouri care folosesc Houdini pentru efecte de lungă durată și proiecte de animație de lung metraj dezvoltă biblioteci de proceduri care pot fi folosite pentru a automatiza generarea multor elemente pentru acel film fără aproape nicio interacțiune cu artistul.De asemenea, unic pentru Houdini este gama de I / O OP-uri disponibile animatorilor, inclusiv dispozitive MIDI, fișiere brute sau conexiuni TCP, dispozitive audio (inclusiv detectarea fonemelor și tonului încorporat), poziția cursorului mouse-ului și așa mai departe. O remarcă deosebită este capacitatea Houdini de a lucra cu audio, incluzând sinteza sunetului și muzicii și instrumentele de procesare a sunetului spațial 3D. Acești operatori există în contextul numit „CHOPs” pentru care Efectele secundare au câștigat un premiu al Academiei pentru realizări tehnice în 2002. VEX (Vector Expression) este unul dintre limbajele interne ale lui Houdini. Este similar cu Renderman Shading Language. Folosind VEX un utilizator poate dezvolta SOP-uri personalizate, POP-uri, shadere etc. Implementarea actuală a VEX utilizează procesarea în stil SIMD [10].

3.2)Unity

Unity este un “game-engine” cross-platform dezvoltat de Unity Technologies, anunțat și lansat pentru prima dată în iunie 2005 la Conferința Mondială a Dezvoltatorilor de la Apple Inc. ca motor exclusiv pentru Mac OS X. De atunci, motorul a fost extins treptat pentru a suporta o varietate de platforme pentru desktop, mobil, consolă și VR. Este deosebit de popular pentru dezvoltarea de jocuri mobile iOS și Android și este utilizat pentru multe dintre jocurile actuale. Este cunoscut pentru a fi ușor de utilizat pentru dezvoltatorii începători și este popular pentru dezvoltarea jocurilor Indie. Motorul poate fi utilizat pentru a crea jocuri tridimensionale (3D) și

bidimensionale (2D), precum și simulări interactive și alte experiențe. Motorul a fost adoptat de industrii din afara jocurilor video, cum ar fi filmul, industria auto, arhitectură, ingineria și construcțiile [12].

Funcții de bază:

- Suport grafic 3D și 2D

Unity acceptă atât grafica 3D, cât și cea 2D - oferind utilizatorului libertatea de a alege stilul de artă pe care îl dorește pentru proiectele lui. Fiecare tip de grafică vine cu propriile instrumente specializate (cum ar fi Sprite Sheet Cutting pentru grafica 2D) și are chiar propriile API-uri de script pe care le puteți desena pentru diferite opțiuni de fizică care sunt potrivite pentru fiecare stil. Grafica 3D oferă, de asemenea, un set extrem de vast de instrumente, precum și posibilitatea de a crea materiale personalizate, de a crea shadere cu Shader Graph, de a regla iluminarea, de a utiliza efecte de post-processing și multe altele. Puteți genera chiar teren 3D sau puteți crea hărți 2D direct în motor, deci există un set de instrumente bine conturate pe care le puteți utiliza pentru orice grafică.

- Arhitectură ușor de înțeles

Unity oferă o metodă foarte transparentă pentru construirea arhitecturii jocului. Fiecare „nivel” dintr-un proiect de Unity este împărțit într-o scenă și fiecare scenă conține toate obiectele de joc de care are nevoie jucătorul pentru nivel - indiferent dacă acesta este fundalul, personajul jucătorului, inamicul, globul sau altceva. Unity oferă, de asemenea, capacitatea de a avea o relație părinte-copil între obiectele din ierarhie, ceea ce face foarte ușoară adăugarea mai multor obiecte (cum ar fi o ținută și o armă) la un obiect cu rol de jucător părinte. În plus, Unity oferă și instrumentul Inspector, care vă oferă acces rapid la toate proprietățile obiectului dvs., ceea ce înseamnă că puteți schimba lucrurile rapid fără a fi nevoie să vă folosiți de cod tot timpul.

- API Unity Scripting

Mai degrabă decât să acționeze orbește, Unity are un API de scriptare puternic care vă oferă acces rapid la funcțiile de care aveți nevoie cel mai des. Aceasta include atât funcții generale de joc, cât și apeluri API specifice care vă permit să accesați anumite caracteristici și nuanțe ale motorului. De exemplu, deși puteți personaliza elementele de interfață din interiorul motorului, cum ar fi culoarea textului, API-ul de scriptare expune și aceste elemente, astfel încât să le puteți

personaliza și din cod. Acest lucru se aplică la tot ceea ce puteți accesa de la Unity Inspector, inclusiv poziția, rotația, materialele, redarea audio și mai mult decât doar cele enumerate. Există, de asemenea, o documentație extinsă pentru a vă ajuta.

- Suport pentru construirea multiplataforma

Jocurile Unity sprijină construirea unui număr imens de platforme. Atâta timp cât se descarcă kitul corespunzător, puteți exporta jocuri pentru Android, iOS, Windows, macOS, Linux, PS4, Xbox One și multe altele. Puteți chiar exporta jocuri HTML5 dacă doriți să vă puneți jocul pe web (presupunând că performanța este optimă). Motorul face, de asemenea, astfel încât să faceți cât mai puține personalizări pentru diferitele versiuni, limitând necesitatea de a avea mai multe versiuni ale proiectului dvs. pentru fiecare platformă.

- Realitate virtuală și capacități de realitate augmentată

Când vine vorba de noile tehnologii VR și AR, Unity este unul dintre principalii susținători pentru dezvoltarea acestor tehnologii. Există numeroase pachete disponibile pentru VR care acceptă aproape toate seturile VR disponibile, se actualizează constant și sunt flexibile cu această tehnologie în schimbare. Puteți chiar să vă testați jocurile VR în motor. AR nu rămâne în urmă, cu numeroase pachete pentru ARCore și ARKit. Unity oferă, de asemenea, AR Foundation, pe care Unity a creat-o pentru a permite dezvoltatorilor să creeze simultan aplicații AR pentru Android și iOS, eliminând necesitatea de proiecte separate.

În plus, Unity are acum și XR Interaction Toolkit, pentru a facilita dezvoltarea jocurilor VR și AR. Deci, este suficient să spunem, Unity este unul dintre cei mai mari susținători ai tehnologiilor XR.

- Magazin

Fie că aveți nevoie de materiale grafice, șabloane pentru un anumit gen de joc, sunet, efecte de particule sau orice altceva, Unity are totul. Magazinul imens oferă o mare varietate de active plătite și gratuite pe care le puteți folosi pentru orice proiect de joc. În timp ce Unity dezvoltă însăși unele dintre acestea, multe sunt create și de comunitate, ceea ce înseamnă că aveți o mare selecție din care să alegeți. Unity face, de asemenea, foarte ușor să adăugați materiale la colecția dvs. și să le instalați în proiect folosind managerul de pachete.

- Pachete dezvoltate de Unity

Similar cu cele de mai sus, Unity însăși oferă gratuit o mulțime de pachete și active proprii care extind funcționalitatea motorului în moduri utile. De exemplu, activul Bolt oferă o modalitate de a implementa scripturi vizuale în motorul Unity. Între timp, Unity Playground oferă un cadru de joc 2D care vă permite să învățați dezvoltarea jocului fără a fi nevoie să codați de la zero. Totul, de la modele gratuite la diferite seturi de jocuri, este oferit gratuit de Unity, oferindu-vă acces rapid la activele aprobate de Unity cu care să vă exersați.

- Opțiuni de redare a conductei

Redarea graficii pe ecran nu este o sarcină ușoară pentru un computer și modul în care se realizează acest lucru poate avea un impact semnificativ asupra performanței jocurilor. Din acest motiv, Unity oferă mai multe opțiuni încorporate pentru conducte de redare pe care le puteți folosi pentru a vă duce jocul de la scenă la ecranul jocului. Acest lucru permite dezvoltatorilor să aleagă conducta de redare care se potrivește cel mai bine proiectelor lor și nevoilor grafice ale proiectelor respective. În plus, Unity oferă, de asemenea, API-ul Scriptable Render Pipeline, care permite dezvoltatorilor să își creeze propria conductă. Deci, există multă libertate în ceea ce privește modul în care jocul este redat jucătorilor dvs.!

- Instrumente de animație

Unity oferă un set divers de instrumente de animație care funcționează atât pentru grafica 3D, cât și pentru grafica 2D. Deși puteți importa cu siguranță animații dintr-un alt program, cum ar fi Blender, Unity vă oferă posibilitatea de a vă anima proiectele direct în cadrul motorului. Aceasta include ajustarea poziției și rotației unui întreg obiect pentru a manipula fizic oasele într-un model 3D. Unity oferă chiar și posibilitatea de a adăuga linii ale oaselor la imaginile 2D. Desigur, toate aceste caracteristici pot fi accesate și prin intermediul API-ului de scriptare, oferindu-vă un control fără precedent asupra modului în care funcționează animațiile dvs.

Mai mult, puteți crea cu ușurință o mașină de stare de animație utilizând sistemul Animator. Aceasta înseamnă nu numai că puteți reda animații pe baza a ceea ce face jucătorul (cum ar fi sărituri), dar puteți face și tranziția dintre fiecare animație adecvată și liniară. Și întrucât animatorul este prezentat într-un stil de diagramă vizuală, este ușor de înțeles cum se leagă totul.

- Adaptabilitate la alte industrii

Deși Unity este în primul rând un motor de joc, este demn de remarcat faptul că Unity a depus, de asemenea, un efort pentru a adăuga caracteristici și adăugiri care fac motorul util pentru alte industrii. De exemplu, Unity este de fapt capabil de a fi utilizat pentru filme de înaltă fidelitate CG datorită opțiunilor sale de canalizare și a instrumentelor de animație, de care mulți realizatori independenți au profitat. De asemenea, Unity a creat, lucruri precum Unity Reflect pentru a oferi dezvoltatorilor de clădiri o modalitate de a-și vizualiza proiectele și de a le conecta la alte programe CAD. Există multe alte exemple, dar Unity este ușor capabil să depășească cu mult scopul inițial pentru care a fost făcut și să ofere suport 3D generalizat în timp real.

- Instrumente de analiză

Pe măsură ce dobândești mai multe abilități ca dezvoltator de jocuri, devine din ce în ce mai important să ai la dispoziție o varietate de instrumente de analiză. Unity oferă mai multe dintre acestea, inclusiv instrumente pentru urmărirea problemelor de performanță și instrumente pentru a observa cu ușurință modul în care jucătorii interacționează cu proiectul tău de joc. În plus, Unity oferă, mai multe modalități de a îmbunătăți depanarea cu aceste instrumente, oferind un mod complex și complet de a înțelege fiecare aspect al jocului tău [13].

2.Prezentarea aplicației

„Taxi Simulator” este creat în Unity[18], încorporând elemente din Houdini.Acesta este un joc care te lasă să experimentezi viața unui șofer de taxi. Dacă îți plac mașinile și îți place să conduci acest joc este pentru tine. Descoperă o lume nouă, dar totodată cunoscută, acțiunea având loc chiar în orașul București. Începi cariera cu o mașină modestă și un salariu mic, dar cu puțină ambiție ajungi să conduci mașini luxoase și să ai un cont bancar considerabil. Vizitează dealership-ul pentru a-ți găsi nouă mașină preferată,care te va ajuta să îți amplifici veniturile.

Acest joc este cu adevărat special. Acesta combină două aspecte foarte importante: jocul de tip simulator și reprezentarea lumii reale. Deși exista jocuri care abordează aceasta idee, nu au un lucru aparte care sa te uimească. Faptul că acțiunea are loc într-un loc real face acest proiect mai mult decât un joc oarecare,oferind jucătorului, pe lângă plăcerea de a conduce, și o plăcere de a explora locuri cunoscute. Poți deveni turist în propriul oraș, faptul că forma străzilor este identică cu cea din realitate fiind un lucru care te poate ajuta și în viața de zi cu zi la a cunoaște mai bine orașul respectiv. Pentru ca în viitor sa fie atras din ce în ce un public mai larg se vor adaugă și alte orașe ale lumii, toate având aerul specific locului și oferind fiecărui jucător un loc familiar.

Jucătorul începe cu un apartament și un garaj pentru mașinile personale.Scena inițială este apartamentul unde acesta își poate salva progresul pe viitor. Odată ce părăsește apartament, jucătorul ajunge în garaj. Pentru a putea să-și înceapă activitatea garajul conține o mașină de început, pe care jucătorul o poate folosi. Jucătorul urcă în mașina dorită și părăsește garajul pentru a ajung în oraș. Aici exista mai multe stații de taxi, amplasate în același loc ca cele din realitate. Conduce la stația dorită și așteaptă ca un client să sosească. În urma cursei acesta primește o suma de bani și experiență care ajută la creșterea nivelului. Jucătorul poate apoi să exploreze alte părți ale orașului și să continuie să facă bani și level pentru a debloca alte lucruri noi.

Scenele jocului:

4.1),,Main Menu’’

- **New Game**

Acesta este un buton care face posibil începutul unui joc nou. Dacă jucătorul decide că dorește ca progresul său sa fie resetat, în momentul în care se apăsă butonul, în folder unde sunt salvate datele despre jucător se introduce un set de datele identice cu cele inițiale.

- **Continue**

Acesta este un buton care face posibil continuarea jocului. În momentul în care se apăsă butonul, folder unde sunt salvate datele despre jucător rămâne nemodificat și se trece la următoarea scenă, jocul încărcând progresul automat acolo.

- **Option**

Acesta este un buton care face posibilă modificarea diferitelor aspecte ale jocului. La apăsare se trece la un meniu diferit care permite jucătorului să facă modificări. Pentru a modifica volumul jocului am creat un slider. Acesta face referire la scriptul ,,OptionMenu’’.

Variabilele acestuia sunt:

-public AudioMixer audioMixer

Acesta este setat manual în scenă făcând referire la mixerul principal.

Funcția care ajută la modificarea volumului este:

-public void SetVolume (float volume):parametrul ,,volume’’ este extras cu din poziția în care slider-ul este setat de jucător. În funcție de mișcarea la stânga sau la dreapta a slider-ului, volumul jocului crește sau descrește.

- **Exit**

Acesta este un buton care face posibil ieșirea din aplicație. În momentul în care se apăsă butonul.

4.2),,Apartment’’

Pentru oferi un mediu cât mai apropiat de viața reală prima interacțiune a jucătorului cu jocul are loc în casă lui. Pentru a încuraja jucătorul să revină aici, principalul scop al casei este de a salva progresul în orice moment dorește jucătorul.

SavePoint

În fiecare scenă există un ,,Player’’ care este setat într-un stadiu default. Pentru a putea păstra progresul dintre scene este nevoie ca la schimbarea scenei să salvăm datele despre caracter pentru a putea apoi să oferim datele schimbate caracterului din scenă următoare.

Scriptul are ca variabile:

- private GameObject player;
- private MoneySystem moneySave;
- private LevelSystem levelSave;
- private LevelSystem expSave;
- private CarCollection garajSave;
- private CurrentCar carNameSave;
- private int triggerCheck = 0;

Funcțiile acestei clase:

- void Start(): Se caută în scenă un obiect cu tag-ul ,,Player’’ și i se atribuie variabilei ,,player’’. Se extrag componentele ,,MoneySystem’’, ,,LevelSystem’’, ,,CarCollection’’și ,,CurrentCar’’ din variabila anterioară și se atribuie variabilelor ,,moneySave’’, ,,levelSave’’ , ,,expSave’’ , ,,garajSave’’ și ,,carNameSave’’.
- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu ,,triggerul", acesta este considerat un ,,Collider". Verificăm tag-ul atribuit parametrului ,,other’’, iar dacă acesta este ,,Player’’ setăm valoarea variabilei ,,triggerCheck’’ la 1.
- private void OnTriggerEnter(Collider other): Această este o funcție specifică unui trigger. În momentul în care un obiect a a parasitcontactul cu ,,triggerul", acesta este considerat un ,,Collider". Verificăm tag-ul atribuit parametrului ,,other’’, iar dacă acesta este ,,Player’’ setăm valoarea variabilei ,,triggerCheck’’ la 0.
- void Update():Dacă ,,triggerCheck’’ este 1 și jucătorul apăsă butonul specific acțiunii, atunci se salvează datele.

4.3),,Garage''

Garajul este o scenă a jocului. Aceasta reprezintă garajul jucătorului, de unde își alege una dintre mașinile deținute pentru a le folosi la job.

SpawnCar

Fiecare mașină care poate fi deținută de către jucător are un loc special în garaj. Acest script este atașat unui obiect oarecare și are rolul de a face ,,spawn'' doar mașinilor deținute de player. Scriptul are ca variabile:

- public string carName;
- public GameObject car;
- private CarCollection carCollection;

Toate variabilele publice sunt setate manual, deoarece fiecare spawn trebuie sa fie personalizat. Funcțiile acestei clase:

- void Start(): Se caută un obiect cu tag-ul ,,Player'' și se atribuie componenta ,,CarCollection'' variabilei ,,carCollection''. Se caută în colecția de mașini a jucătorului și în cazul în care mașina cu numele ,,carName'' se regăsește, atunci se instantiaza obiectul ,,car''.

4.4),,World''

4.4.1)Canvas

,,Canvas''-ul este o ,,pânză'' unde se găsesc elementele care țin de interfața cu utilizatorul. Acesta ajută la a oferi informații direct pe ecran. Elementele care se găsesc în acest joc sunt:

4.4.1.1)Info

Cu ajutorul unui script extragem din componentele ,,Player-ului'' informații despre : level, experiența și suma de bani deținută.

4.4.1.2)MiniMap

Cu ajutorul unei camere atașată deasupra ,,arrow''-ului am setat un ,,Render Texture'' care preia imaginile. Aceste imagini sunt apoi redate pe ecran cu ajutorul unui element ,,UI''.

4.4.1.3)Gps

Și pentru GPS ,am atașat o camere ,,arrow''-ului într-un unghi specific și am setat un ,,Render Texture'' care preia imaginile. Aceste imagini sunt apoi redate pe ecran cu ajutorul unui element ,,UI''.

4.4.1.4)Vitezometru

Pentru a putea oferi jucătorului informații despre viteză de deplasare am făcut un script pe care l-am atașat unui element ,,UI'' de tip ,,Text''.

Script-ul se numește ,,Vitezometru'' și are ca variabile:

- private Rigidbody car;
- public Text SpeedText;

Funcțiile clasei sunt:

- void Update(): Se caută un obiect cu tag-ul ,,PlayerCar'' și i se atribuie componenta ,,Rigidbody'' variabilei ,,car''. Se extrage viteza mașinii în variabila locală ,,carSpeed'' cu ajutorul funcției ,,velocity.magnitude()''. Apoi variabila este transformată în tip string și atribuită variabilei ,,text'' care reprezinta componenta ,,UI'' a vitezei.

4.4.2)Gps

Scopul acestui obiect este de a simula un GPS, oferind astfel un sentiment al realității. Are scopul de a fi folosit ca o aplicație a telefonului mobil din joc, asemănătoare cu ,,Waze''.

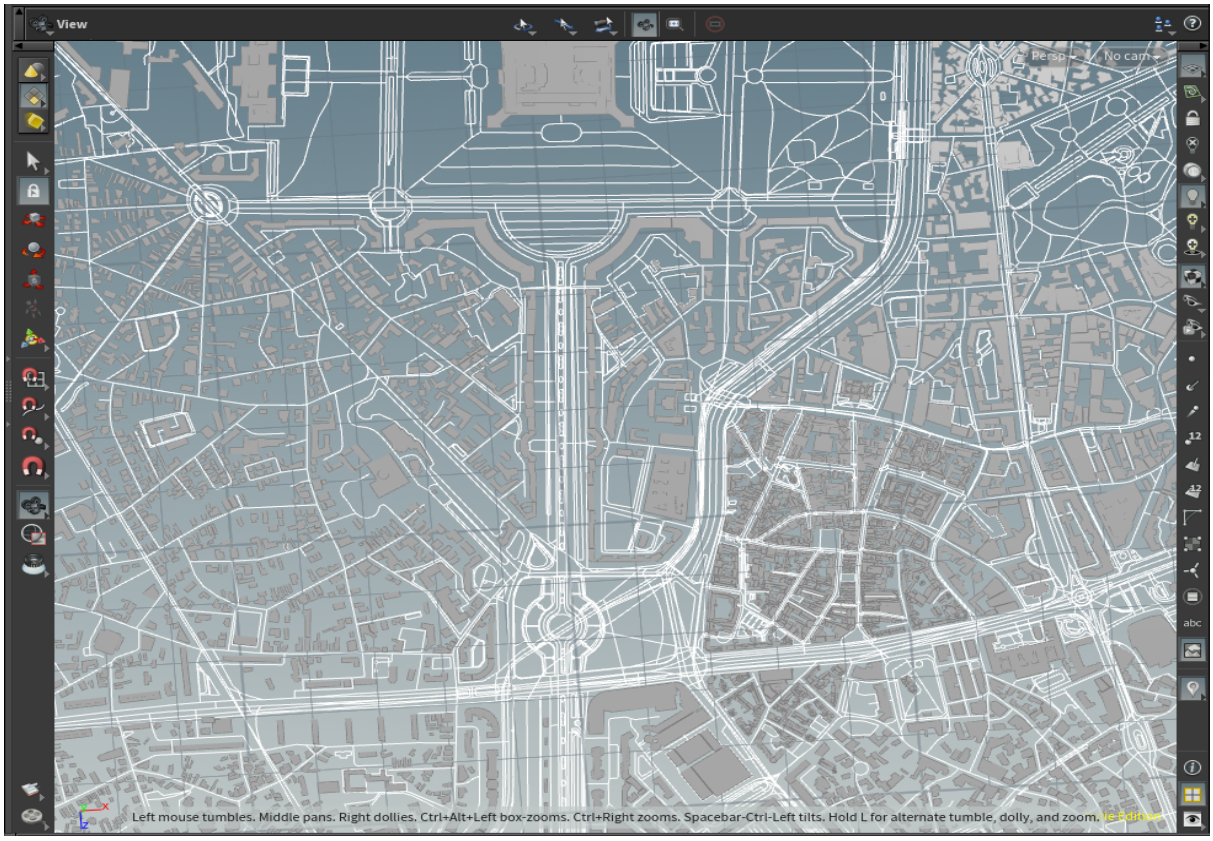
4.4.2.1)Mapa

Mapa jocului este realizata în HoudinFx[9] și este facută procedural. Pentru a realiza bazele orașului am făcut 2 proiecte separate pentru: străzi și clădiri.

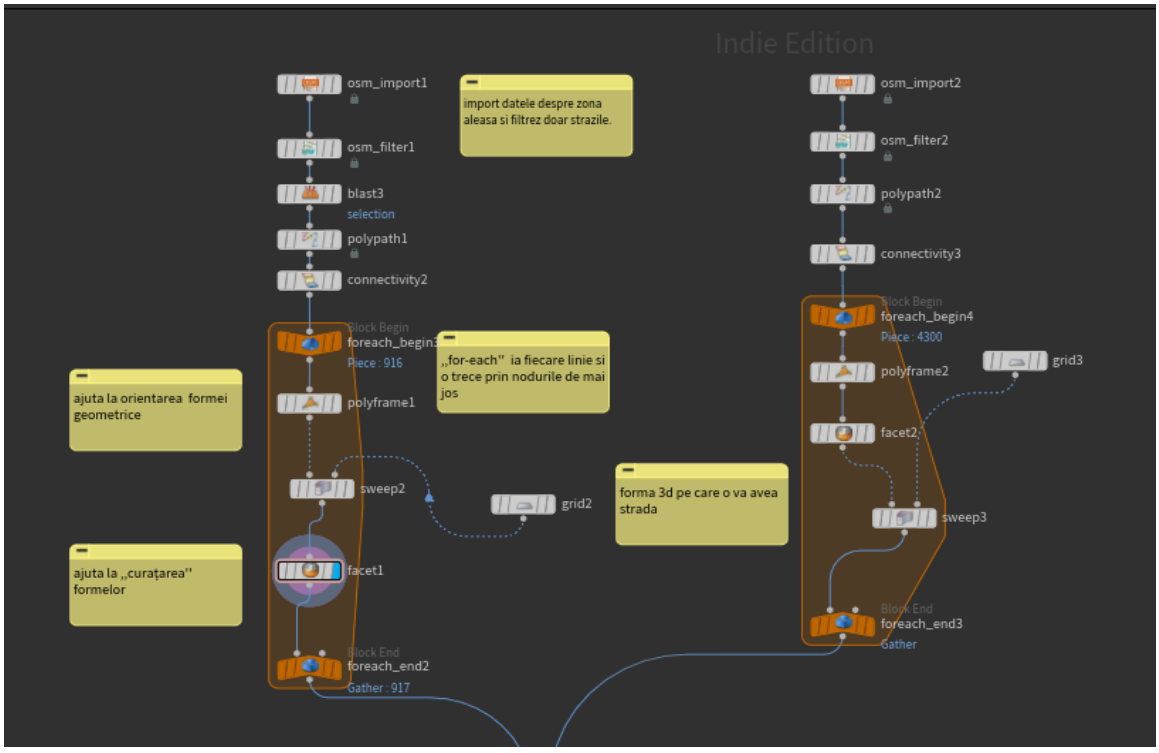
Pentru ambele proiecte am extras informații referitoare la tipul de străzi, tipul de clădiri și amplasarea tuturor de pe ,,www.openstreetmap.org''.

- **Strazi:**

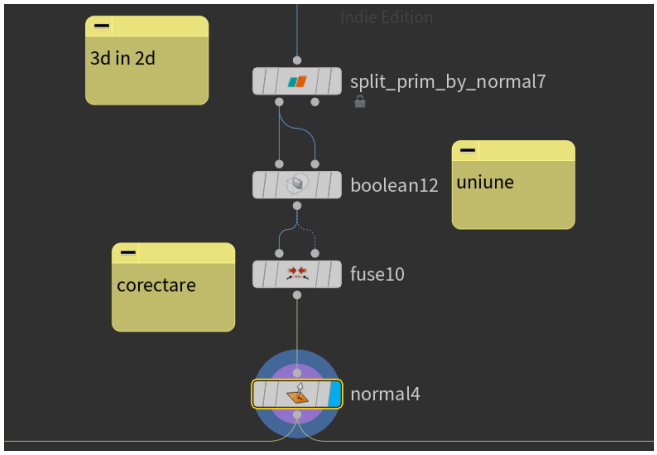
-Cu ajutorul unui tool am încarcat informațiile extrase în HoudiniFx.În acest stadiu străzile sunt reprezentate doar ca niște linii.



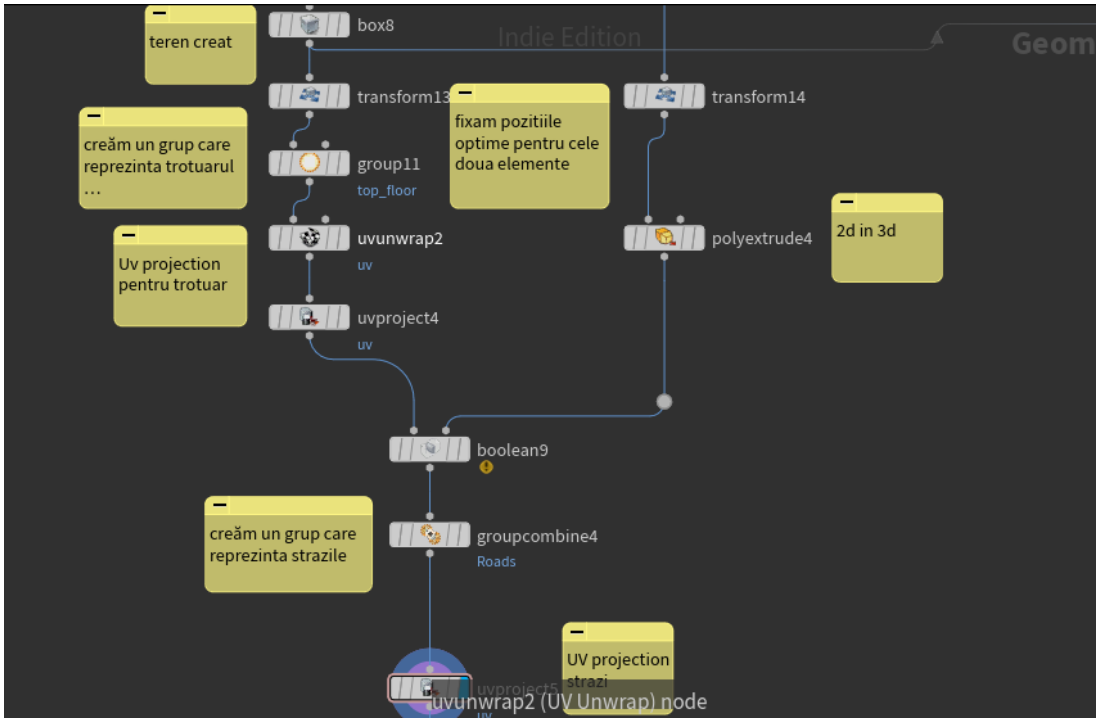
-Am folosit un block de noduri ,,For-Each'' pentru a lua fiecare linie și a o transforma într-o forma 3d, de lățime variabile, în funcție de tipul de stradă reprezentat.



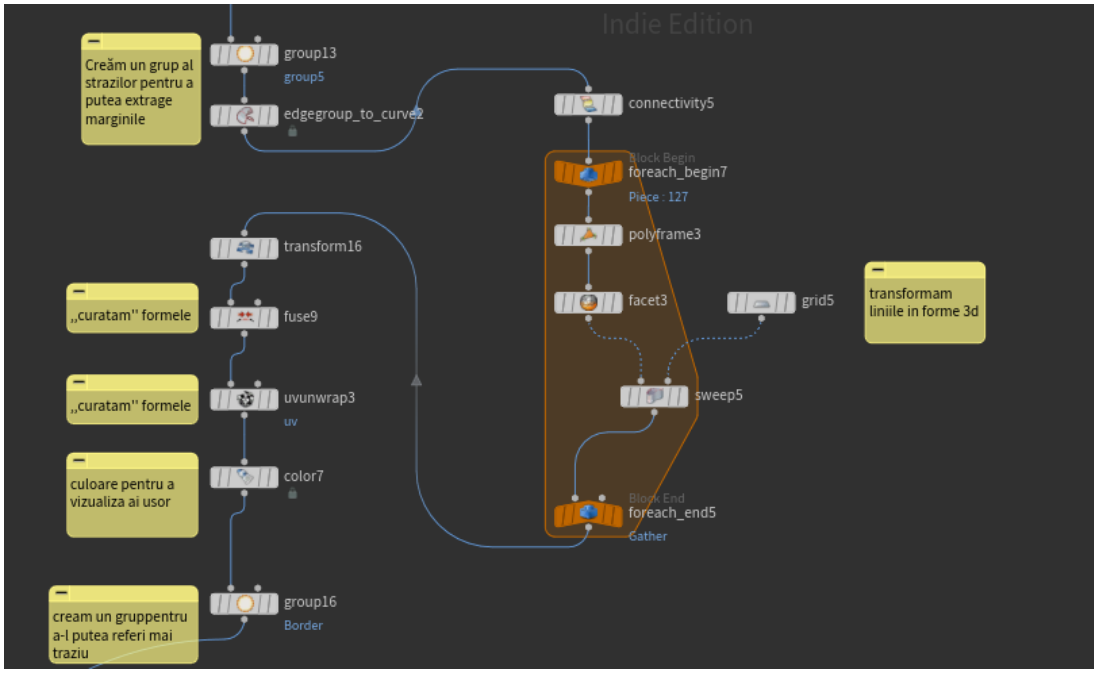
-Deoarece am transformat linii în obiecte 3d este de așteptat ca mai multe dintre acestea sa se intersecteze în anumite puncte. Acest lucru poate creea probleme în viitor atat de natură estetică , cât și de functionalitate. Transformăm forma 3d într-o forma 2d și facem o uniune a formelor geometrice, obținând astfel din două sau mai multe forme care se suprapun,o formă care le unește pe toate, partea comună fiind zona de legatură. Deoarece dacă un unghi dintre 2 linii este prea mare,atunci nu vom avea 2 forme care sa formeze un plan continuu, avem nevoie să corectăm acest lucru cu un nod care creează o legătură între două forme geometrice care sunt la o distantă relativ mică, aleasă de utilizator.



-Pentru a reprezenta trotuarul creăm un paralelipiped suficient de mare cat sa cuprindă suprafața pe care se desfășoară străzile. Îi atribuim un grup ca să putem în viitor sa referim doar aceasta parte a modelului. Pentru a putea sa îi atașăm diferite materiale în Unity este nevoie sa setam un ,,UV projection''(Cum funcționează și în viața reala ochiul uman poate vedea culorile deoarece fiecare obiect respinge razele de lumina ale soarelui înapoi la ochiul nostru,dar trebuie să setăm modul în care se resping). Transformăm modelul 2d al străzilor făcut anterior într-un model 3d pentru a putea face o operație boolean de tip diferență între terenul creat și străzi. În urma acestui proces pe teren rămâne o ,,amprentă'' a formei străzilor în teren.



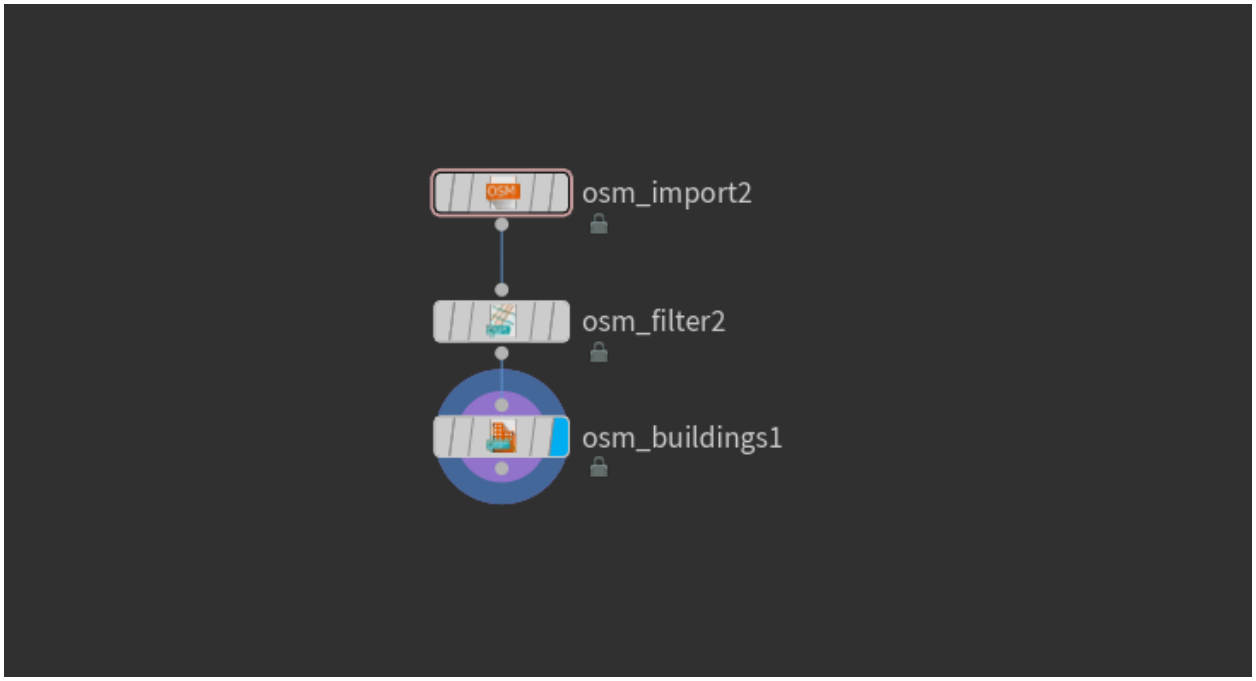
-Cu forma 2d a străzilor aflată în pașii anteriori, selectam marginile. Acestea fiind tot linii le transformăm în obiecte 3d după același algoritm ca la transformarea străzilor, cu o forma specifica bordurilor. Este și în acest caz nevoie de a adăuga un „UV Projection''.



-În ultimul pas combinăm toate modele de mai sus într-unul singur și setăm materialele aferente fiecărui grup.

- **Cladiri:**

Pentru ca nu este nevoie de un nivel de detaliu al cladirilor, acestea fiind doar o reprezentare 3d procesul este destul de simplu. Extragem datele despre zona selectata si filtram doar forma cladirilor. Le facem apoi modelul 3d cu ajutorul unui tool pus la dispoziție de program.



4.4.2.2)MapControl

Pentru a putea naviga mapa, avem nevoie de script-ul ,,MapControl''. Acesta ne permite să mergem în cele 4 direcții și să mărim sau micșorăm locul dorit.Script-ul se atașează unei camere.

Variabilele script-ului sunt:

- public float panSpeed = 20f;
- public float panBorderThickness = 10f;
- public Transform player;
- public Vector2 panLimitx;
- public Vector2 panLimitz;
- public float scrollSpeed = 20f;
- public float minY=20;
- public float maxY=300;

Toate variabilele publice au fost setate manual în scenă.

Funcțiile atașate sunt:

- private void Start():Pentru ca poziția de început a camerei sa coincidă cu poziția unde se află săgeata jucătorului, copiem valorile componentei ,,Transfrom'' a variabilei ,,player''.
- void Update(): În funcție de tasta apăsată modificăm poziția camerei în limitele ,,panLimitx'' și ,,panLimitZ''. De asemenea cu ,,wheelscrool''-ul mouse-ului se realizează ,,zoom in'' și ,,zoom out'' în limitele ,,minY'' și ,,maxX''.

4.4.2.3)Arrow

Pe harta GPS-ului, jucătorul este reprezentat ca o săgeată. Pentru ca aceasta să poată arată poziția jucătorului și pentru a putea să ofere o ruta de la poziția acestuia la punctul de interes, avem nevoie de 2 scripturi:

FollowPlayer

Acest script se atașează obiectului ,,Arrow'' și are ca scop sa arate pe minimapa locul corespunzător cu poziția jucătorului.

Variabilele script-ului sunt:

- private Transform player;
- private Transform playerCar;

Funcțiile atașate sunt:

- void Update(): Poziția jucătorului este urmărită atât în timpul cât folosește mașina dar și cand o părăsește. La fiecare frame se caută obiectele cu tag-ul ,,Player'' și ,,PlayerCar''. Dacă este găsit obiectul cu tag-ul ,,Player'' atunci copiem valorile componentei ,,Transform'' a acestuia și le atribuim săgeții, altfel copiem componenta din ,,PlayerCar''.

ArrowController

Acest script se atașează obiectului ,,Arrow'' și are ca scop să ofere jucătorului posibilitatea de a selecta un punct pe mapa unde dorește sa ajungă să îi fie indicată o rută în timp real. Pentru a functiona este nevoie ca terenului să îi fie atribuita o componenta ,,NavMesh''.

Variabilele script-ului sunt:

- public GameObject MapCam;
- private Camera cam;
- public NavMeshAgent agent;
- private LineRenderer line;
- private List<Vector3> point;
- private int ok = 0;
- private RaycastHit lastHit;
- private Ray ray;
- private RaycastHit hit;

Funcțiile atașate sunt:

- void Start(): Selectam componentele ,,NavMeshAgent'' și ,, LineRender'' atașate obiectului implicit, și le atribuim variabilelor ,,agent'' și ,,line''.
- void Update(): Se verifica dacă jucătorul a apăsat pe ecran cu ajutorul variabilei ,,ray'' care primește input de la mouse. Punctul selectat este reținut în variabila ,,ray''. Setăm destinația la punct cu ajutorul variabilei ,,agent'' și cu ajutorul funcției ,,DisplayLineDestination()'' se trasează o linie colorată de la poziția curentă la destinație. Pentru a oferi jucătorului posibilitatea de a anula ruta către o destinație, dacă ,,hit'' este același cu ,,lastHit'', atunci ruta dispare. De asemenea dacă poziția obiectului implicit ajunge la o distanță relativ mică de punctul final, atunci se consideră că s-a ajuns la

destinație și ruta dispare. Deoarece jucătorul trebuie să se deplaseze pe carosabil, ruta dată nu trece prin parcări, spațiu verde, trotuar etc. Cu toate acestea jucătorul are libertatea să facă acest lucru, iar cu ajutorul funcției ,,IsAgentOnNavMesh()'' verificăm dacă acesta se află pe carosabil sau nu. De fiecare dată când jucătorul părăsește și revine pe carosabil, ruta este recalculată automat.

- public bool IsAgentOnNavMesh(GameObject agentObject):Această funcție verifică poziția jucătorului. Dacă aceasta se află pe carosabil atunci întoarce ,,true'', altfel se va întoarce ,,false''.
- private void DisplayLineDestination(): Aceasta funcție trasează o linie între punctele care alcătuiesc traseul ,,agent''-ului.

CameraSwitch

Acest script se atașează obiectului ,,dummy'' și are ca scop să schimbe camera activă dintre camera ,,Player''-ului și camera GPS cu ajutorul căreia setăm locul unde vrem să ajungem.

Variabilele script-ului sunt:

- public GameObject cam1;
- public GameObject camGps;
- public GameObject car;
- public GameObject player;
- public Transform PlayerGpsArrowCam;
- public Transform PlayerGpsArrow;
- private Camera camera1;
- private Camera camera2;

Funcțiile atașate sunt:

- void Update():Dacă jucătorul a apăsat tasta corespunzătoare, atunci se face schimbarea între camere. Dacă variabila ,,camGps'' este activa, atunci ,,cam1'' devine inactivă, deasemena si ,,car'' și ,,player''. Pentru a avea camera în poziția potrivită setăm poziția camerei GPS-ului la poziția ,,săgeții'' care reprezintă jucătorul. Dacă ,,cam1'' devine activă, atunci și celelalte obiecte dezactivate mai sus sunt reactivate.

4.4.3)Oras

4.4.3.1)Mapa

Mapa jocului este realizata în HoudinFx și este facută procedural. Pentru a realiza bazele orașului am făcut 2 proiecte separate pentru: străzi, clădiri și props .Pentru ambele proiecte am extras informații referitoare la tipul de străzi, tipul de clădiri și amplasarea tuturor de pe „www.openstreetmap.org”.

- **Strazi:**

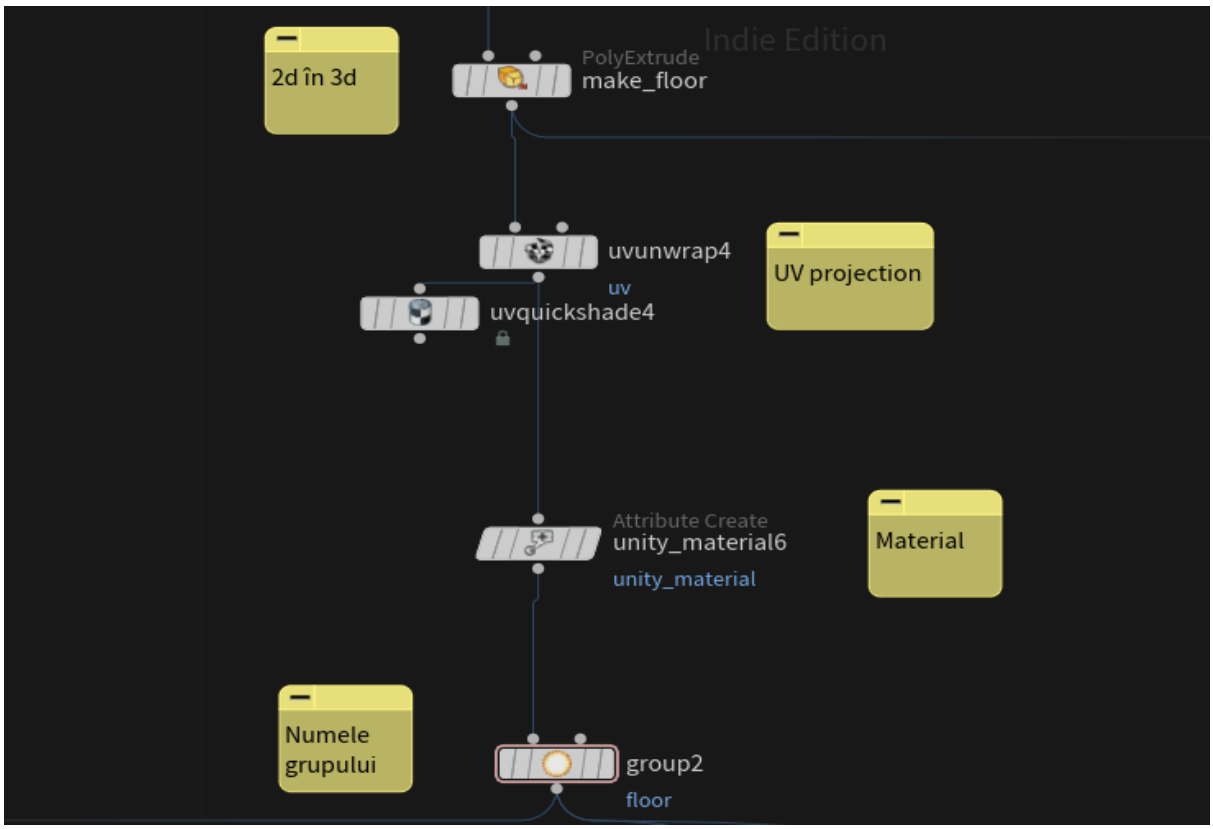
Este același proces ca cel de la mapa Gps-ulu.

- **Clădiri**

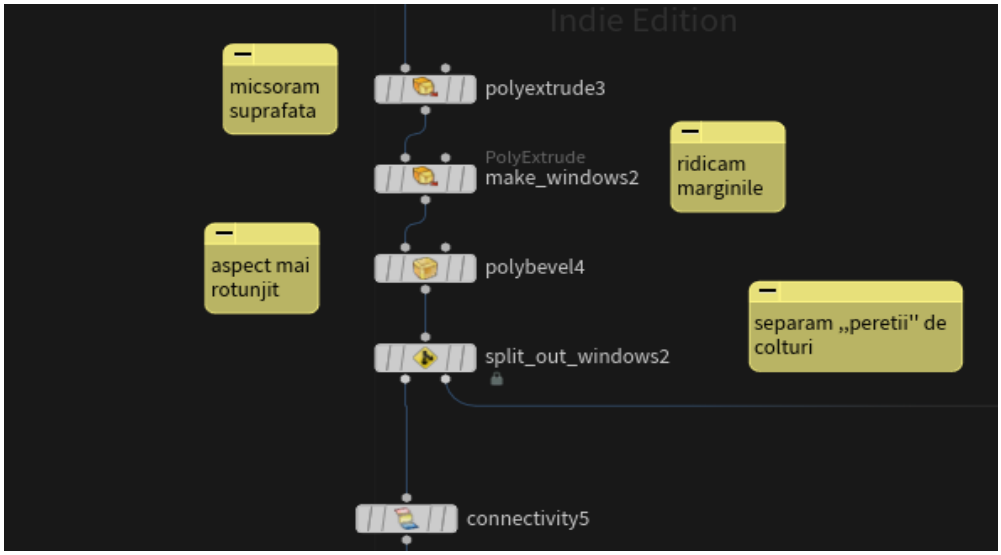
Folosim același proces de mai sus pentru a extrage și a filtra doar formele cladirilor.

-Parter

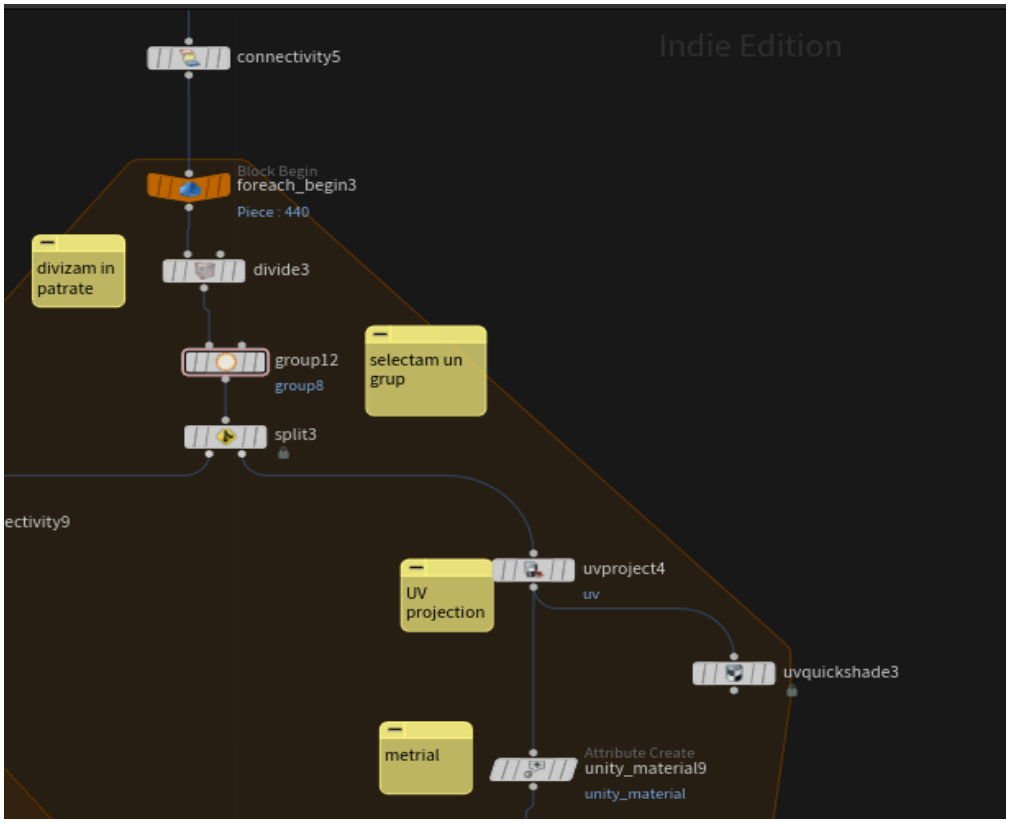
Începem prin a face podeaua nivelului. Transformăm formele 2d în forme 3d. Adaugăm si „UV projection” pentru a putea atribui materiale.



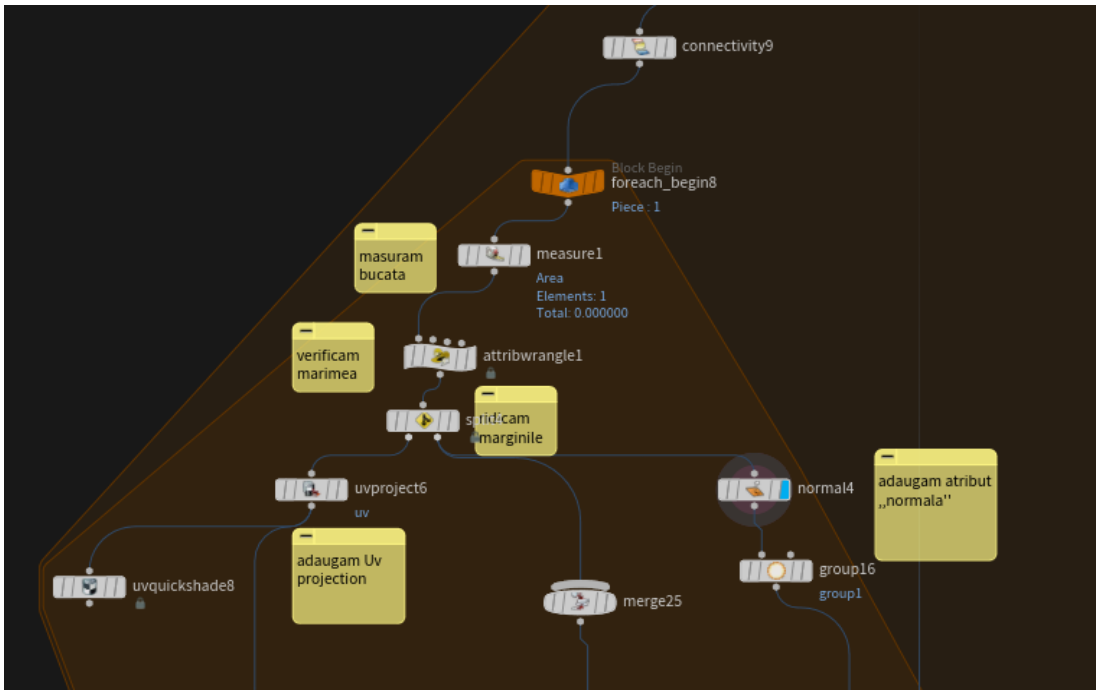
Micșorăm suprafața și înălțăm marginile, creând astfel înălțimea pereților.

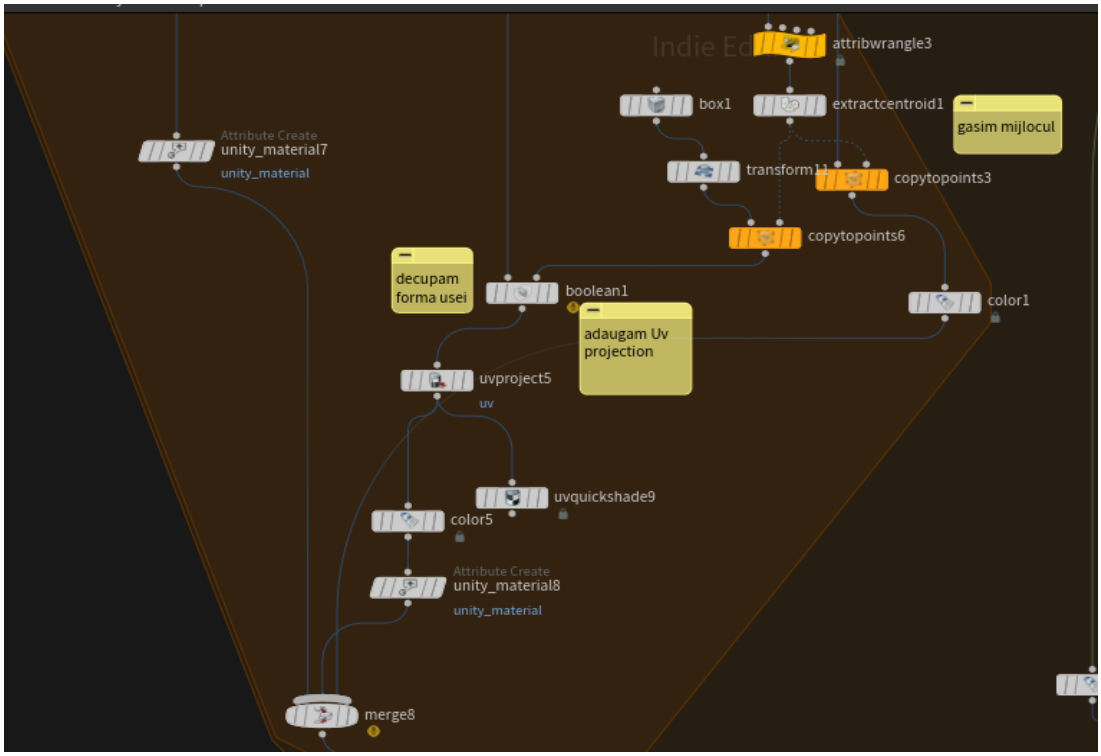


Folosim un block „For-Each'' pentru a selecta fiecare bucata întreaga. Divizam fiecare bucata în dreptunghiuri relativ egale între ele. Selectam o bucata din fiecare grup de zece bucăți. Celor 9 bucăți doar li se atribuie un material cu ajutorul unui „UV projection''.



Dacă bucata aleasă precedent are o dimensiune corectă și nu este o geometrie creată doar pentru a ajutat la obținerea celorlalte, atunci o folosim. Găsim mijlocul acesteia și îi atasam obiectul ce reprezinta usa. Cu ajutorul unei operații boolean de tip diferența ,,decupam'' forma ușii din ,,perete''. La sfarsit adaugam ,,UV Projection'' atat bucăților selectate ,cat și celor neselectate.



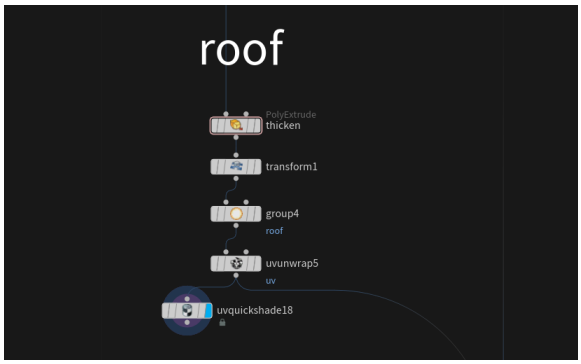


-Etajele cu ferestre

Se obțin printr-un proces similar cu cel al Parterului, înse se adauga ferestre.

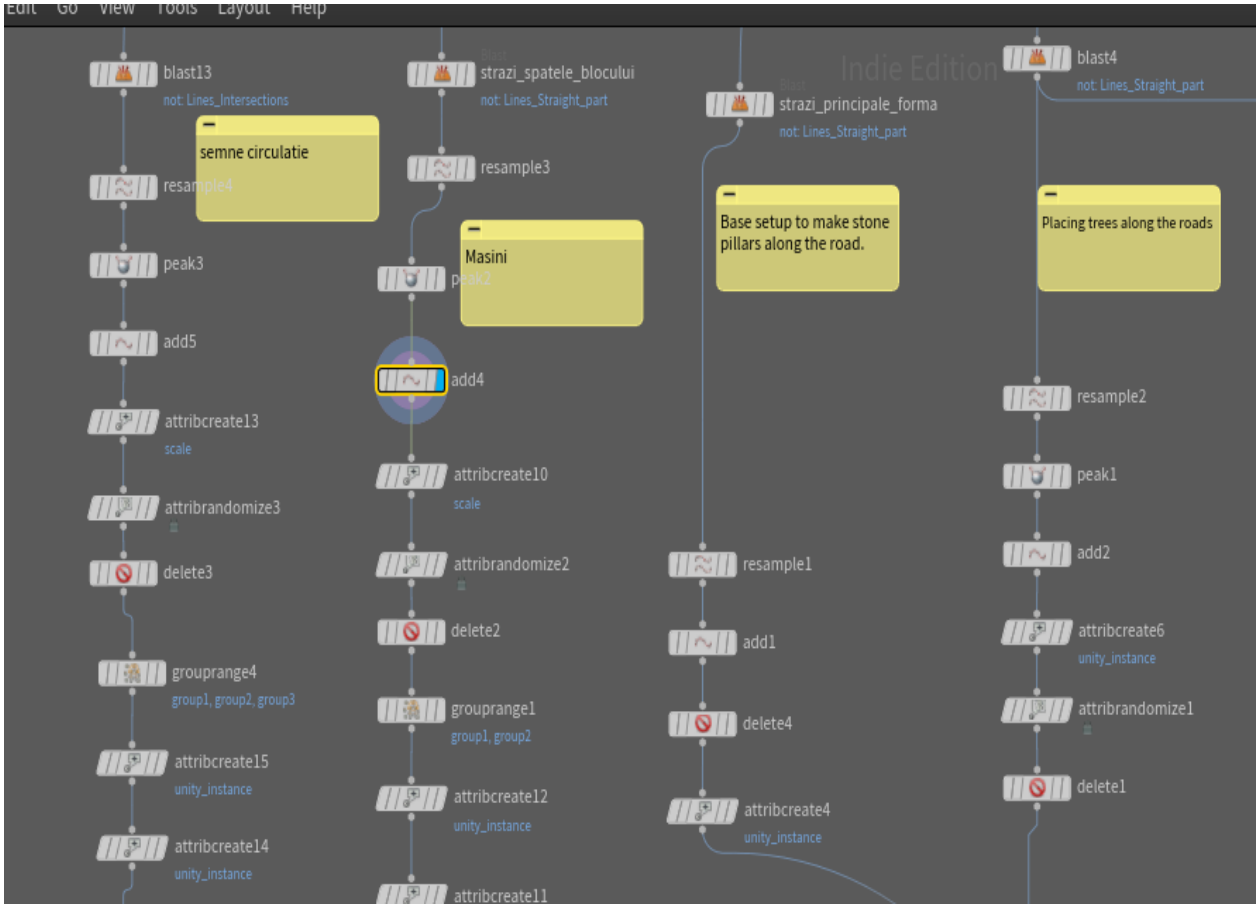
-Acoperiș

Selectăm forma clădirii și ridicăm în trepte înălțimea.



- **Props**

Selectam marginile străzilor realizate mai sus. Luăm porțiuni, la întâmplare, din aceste linii și de-a lungul lor adăugat puncte. Ștergem puncte în mod aleator, pentru a simula pozițiile neuniforme ale lucrurilor din realitate. Formăm mai multe grupuri din punctele rămase , cărora le atribuim diferite obiecte [14].



4.4.3.2)Player-ul

Acesta reprezinta caracterul jucatorului.

Player Controller

Este nevoie de un script care este responsabil pentru controlul ,,Player-ului'' pe tot parcursul jocului. Pentru a realiza acest lucru am folosit ,,Invector Third Person Controller Asset'' [16].

Money System

Jocurile utilizează o mare varietate de monede online. Aceste monede pot avea nume diferite, dar vor fi utilizate pentru același scop final. ,,Taxi Simulator'' folosește un sistem bazat pe misiuni, sub forma unor curse de taxi, pentru a permite jucătorilor să câștige bani. Un factor comun în toate jocurile, inclusiv în acesta, este că moneda jocului ajută la îmbunătățirea caracterului. Spre exemplu, cu banii strânși jucătorul poate cumpara noi mașini, dorința de a-și completa colecția de masini sau de a cumpara masina mult dorința ținându-l în joc. Pe lângă

asta, cu cat mașina cumpărata este mai scumpă,cu atat bonusul pentru cursele îndeplinite cu ea este mai mare, deci jucătorul din dorința de a evolua va fi nevoit sa își îmbunătățească mașina.

Tot acest sistem ajută, din punct de vedere al dezvoltatorului, la atragerea și păstrarea unui jucător. În mod natural, suntem programați să vrem să avem realizări, să simțim că evoluăm, să simțim că ne pricepem la un lucru. Din păcate, creierul nu face diferența dintre ce este real și ce este adevărat, așa că reușitele din joc ne alimentează și ne fac să ne simțim împliniți. Dacă jocul nu ar avea nimic de cumpărat, nu ar avea un scop asemanator cu viața reală, interesul jucătorului s-ar pierde. Prezența unei monezi asigură atât sentimentul de împlinire în obținerea ei,cât și sentimentul de împlinire prin lucrurile care sunt deblocate în schimbul acesteia. Sentimentul de a aduna resurse este unul de baza, așa că, deși poate dintr-un punct al jocului venitul depășește cu mult cheltuielile și există un exces, jucătorul tot se bucură că are un număr mare de monezi.

Scriptul adăugat Player-ului se numește ,, Money System” și are ca variabile:

- private int money

Funcțiile acestei clase:

- public int GetMoney(): Deoarece variabila ,,money’’ este private, este nevoie de o funcție care să transmită valoarea acesteia.
- public void AddMoney(int suma): Deoarece variabila ,,money’’ este private, este nevoie de o funcție care să modifice valoarea acesteia. Funcția are un parametru ,,suma’’care se adaugă la valoarea actuala.
- public void SubstractMoney(int suma): Deoarece variabila ,,money’’ este private, este nevoie de o funcție care să modifice valoarea acesteia. Funcția are un parametru ,,suma’’care se scade din valoarea actuala.

Level System

Un ,,Level System” funcționează și ca un mod de a echilibra bonusurile jucătorului. Dacă ,spre exemplu vrem să introducem o mașină care oferă un bonus foarte mare pentru bani, dacă aceasta este blocată la un anumit nivel, atunci nu trebuie să ne mai facem griji cu privire la echilibrarea prețurilor pentru a obține un anumit ritm al ,,gameplay’’-ului bun.

Din punctul de vedere al jucătorului, existența unui level oferă o modalitate de măsurare ușor de înțeles a progresului lui. Există, de asemenea, psihologia obținerii recompenselor nivelului ca o formă de motivare a jucătorului să continue. Pentru a continua să se joace, jucătorul trebuie să

fie capabil să descopere lucruri noi sau sa schimbe lucrurile deja prezentate. Acesta este motivat sa joace jocul prin recompense. Spre exemplu , fiecare mașină nouă, pe lângă suma de bani necesară pentru a o cumpara, este deblocată la un anumit nivel, în acest mod jucătorul simțind ca face un progres și are un scop. În plus, odată cu creșterea în level, jucătorul obține și anumite avantaje, cum ar fi un boost al banilor câștigați.

Scriptul adăugat Player-ului se numește ,, Level System" și are ca variabile:

- private int level
- private int experience

Funcțiile acestei clase:

- void Update(): În această clasă verific daca experiența acumulată de către player este mai mare decât cea necesară pentru a trece la level-ul următor. Dacă experiența este suficientă, se scade cantitatea necesară și nivelul jucătorului este actualizat.
- public int GetLevel: Deoarece variabila ,,level'' este private, este nevoie de o funcție care să transmită valoarea acesteia.
- public int GetExp(): Deoarece variabila ,,experience'' este private, este nevoie de o funcție care să transmită valoarea acesteia.
- public void AddExp(float amount): Deoarece variabila ,,level'' este private, este nevoie de o funcție care să modifice valoarea acesteia. Functia are un parametru ,,amount'' care reprezintă valoarea brută care trebuie adăugată la experiență. În interiorul funcției , din ,,experience''este scăzută o valoare direct proporțională cu nivelul.

CurrentCar

Acestui script face parte din elementele care se rețin în urma salvării progresului. Acest script memorează numele mașinii pe care player-ul o conduce în fiecare momentul. Pentru a putea face trecerea din scena ,,Garage'' în scena ,,World '' și a putea transmite mașina pe care player-ul a decis sa o folosească, avem nevoie de acest script pentru a reține alegerea lui și a-i oferi mașina corespunzătoare.

Scriptul adăugat Player-ului se numește ,, CurrentCar" și are ca variabile:

- private string currentCarDriven

Funcțiile acestei clase:

- public string GetCurrentCar() : Deoarece variabila ,,currentCarDriven'' este private, este nevoie de o funcție care să transmită valoarea acesteia.
- public int GetCurrentCar(): Deoarece variabila ,,currentCarDriven'' este private, este nevoie de o funcție care să transmită valoarea acesteia.

CarCollection

Acest script reține mașinile pe care player-ul le-a achiziționat. Acest script ajută alte două mecanici ale jocului. În momentul în care player-ul încearcă să cumpere o mașină este verificat ca mașina sa nu apară ca duplicat în colecția acestuia. De asemenea când i se face trecerea în garaj, pentru a putea face ,,spawn'' doar mașinilor deținute, scriptul verifica dacă mașina respectivă se află în colecția player-ului.

Scriptul adăugat Player-ului se numește ,, CarCollection'' și are ca variabile:

- private List<string> garaj = new List<string>();

Funcțiile acestei clase:

- public int SearchCar(string carName): Se cauta în lista ,,garaj'' dacă există o mașină cu numele parametrului ,,CarName'' și în caz afirmativ se returnează 1, altfel se returnează 0
- public void AddCar(string carName): Deoarece variabila ,,garaj'' este private, este nevoie de o funcție care să adauge acestei listei elementul carName.

4.4.3.3)PlayerCar

Reprezintă mașina controlată de jucător.

CarController

Este un script responsabil pentru controlul ,,PlayerCar-ului'' pe tot parcursul jocului. Pentru a realiza acest lucru am folosit ,,Standard Assets CarController'' [17].

CarLights

Pentru o experiență mai reală a jocului am adăugat și un mod pentru jucător de a controla luminile pentru : semnalizarea, avarii , faza lungă , faza scurta. Și de asemenea am conectat acționarea frânei de către jucător cu luminile aferente.

Scriptul adăugat PlayerCar-ului se numește ,, CarLights'' și are ca variabile:

- private float blinkDurationL=0;
- private float blinkDurationR=0;
- private float blinkDurationH=0;
- private float lastBlinkL=0;
- private float lastBlinkR=0;
- private float lastBlinkH=0;
- public float turnTimer=0;
- public float hazardTimer=0;
- public Renderer brakeLights;
- public Material brakeLightOn;
- public Material brakeLightOff;
- public Renderer turnLightL;
- public Renderer turnLightR;
- public Material turnLightOn;
- public Material turnLightOff;
- public Renderer hazardLights;
- public Material hazardLightOn;
- public Material hazardLightOff;
- public GameObject fazaScurtaL;
- public GameObject fazaScurtaR;
- public GameObject fazaLungaL;
- public GameObject fazaLungaR;

Funcțiile acestei clase sunt:

- void Update(): această funcție apelează toate celelalte funcții.
- public void Brake():inițializăm variabila locală ,,brakePoz'' cu 0. De asemenea reținem în variabila ,,mats'' materialele folosite în renderul ,,brakeLights''. Căutăm poziția elementului din lista de materiale ,,mats'' care corespunde cu materialul ,,brakeLightOff'' și salvăm în ,,brakePoz'' poziția acestuia. Dacă jucătorul acționează tasta corespunzătoare frânei, atunci modificăm materialul de pe poziția ,,berakePoz'' în materialul ,,breakLightOn'' ,care are un material cu emisie roșie,oferind astfel iluzia dorită. Când

jucătorul eliberează frâna modificăm înapoi materialul de pe poziția ,,brakePoz'' în materialul ,,breakLightOff''.

- public void turnSignal():fiecare semnalizare este tratată separat,deoarece sunt activate individual. Pentru semnalizarea stangă:inițializăm variabila locală ,,turnPozL'' cu 0. De asemenea reținem într-o variabila ,,matsL'' materialele folosite în renderul ,,turnLightL''. Căutăm poziția elementului din lista de materiale ,,matsL'' care corespunde cu materialul ,,turnLightOff'' și salvăm în ,,turnPoz'' poziția acestuia.Când jucătorul acționează butonul semnalului stâng oprim semnalul drept. Dacă semnalul stâng este deja activ atunci îl oprim,iar în caz contrat îl pornim.Pentru a simula lumina intermitenta a semnalizarii, schimbăm la un interval de 0.5 secunde materialul din ,,turnLightOff'' în ,,turnLightOn'' și invers, cat timp este activ. Pentru semnalizarea dreapta este același proces de mai sus ,în oglindă.
- public void hazardLights():inițializăm variabila locală ,,hazardPoz'' cu 0. De asemenea reținem într-o variabila ,,matsH'' materialele folosite în renderul ,,hazardLights''. Căutăm poziția elementului din lista de materiale ,,matsH'' care corespunde cu materialul ,,hazardLightOff'' și salvăm în ,,hazardPoz'' poziția acestuia.Când jucătorul acționează butonul pentru avarii, dacă aceste sunt deja active atunci le oprim,iar în caz contrat le pornim. Pentru a simula lumina intermitenta a avariilor, schimbăm la un interval de 0.5 secunde materialul din ,,hazardLightOff'' în ,,hazardLightOn'' și invers, cat timp este activ.
- public void fazaScurta():Când jucătorul acționează butonul pentru faza scurta, oprim faza lunga.Dacă faza scurta este deja activă atunci o oprim,iar în caz contrat o pornim.
- public void fazaLunga():Când jucătorul acționează butonul pentru faza lunga, oprim faza scurta.Dacă faza scurta este deja activă atunci o oprim,iar în caz contrat o pornim.

CarName

Fiecare tip de mașină are un nume unic. Acest script reține numele mașinii pentru ca mai târziu să poată fi identificată, spre exemplu de scripturi ale player-ului precum CurrentCar(). Scriptul adăugat PlayerCar-ului se numește ,, CarName" și are ca variabile:

- public string carName;

Aceasta este o variabilă publică pentru a putea fi modificată pentru fiecare masina în parte.

Bonus

Fiecare mașină oferă player-ului un anumit bonus pentru suma de bani câștigată și pentru experiența primită în urma unei curse. Pentru a stimula player-ul să evolueze și să continuie să joace jocul, bonusurile cresc odată cu cererile pentru mașina cumpărată.

Scriptul adăugat PlayerCar-ului se numește ,, Bonus" și are ca variabile:

- public int expBonus;
- public int moneyBonus;

Aceste variabile sunt setate manual pentru fiecare mașină nou adăugată.

Pe lânga script-urile prezentate mai sus, **PlayerCar** mai are ca ,,children''și alte doua obiecte. Pentru ca **Player**-ul sa poată interacționa cu mașina, este nevoie de 2 script-uri: EnterCar și ExitCar.

EnterCar

Acest script este atribuit unui obiect cu funcția de trigger. Cu ajutorul acestuia este posibil sa simulam îmbarcarea player-ului în masina.

Scriptul adăugat se numește ,,EnterCar" și are ca variabile:

- public GameObject carCam;
- private GameObject playerCam;
- private GameObject thePlayer;
- public GameObject exitTrigger;
- public GameObject theCar;
- private CurrentCar currentCar;
- public CarName carName;
- private int triggerCheck=0;

Toate variabilele publice sunt setate manual pentru fiecare mașina căruia i se atribuie scriptul.

Funcțiile acestei clase:

- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu triggerul, acesta este considerat un ,,Collider''. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,Player'', atribuim variabilei ,,triggerCheck'' valoarea 1.
- private void OnTriggerExit(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un obiect care a intrat în contact cu triggerul înainte și oprește

coliziunea acesta este considerat un ,,Collider''. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,Player'', atribuim variabilei ,,triggerCheck'' valoarea 0.

- void Start(): Sunt dezactivate scripturile ,,CarController'', ,,CarUserControler'', ,,CarAudio''și CarLights'' pentru a nu fi controlată mașina odată cu player-ul. De asemenea si ,,exitTrigger'' este dezactivat deoarece nu ar avea sens ca player-ul să poată ieși din mașină dacă acesta încă nu a intrat.
- void Update(): Dacă ,,triggerCheck'' are valoarea 1, înseamna că player-ul este în dreptul ușii și poate intra. Se caută obiectele cu tag-ul ,,Player'' si ,,MainCamera'' pentru a li se atribui variabilelor ,,thePlayer'' și ,,playerCam'', iar variabilei ,,currentCar'' i se atribuie componenta ,,CurrentCar'' a obiectului ,,thePlayer''. Următorul pas este să verificăm dacă Player-ul a apăsat butonul care corespunde cu intrarea în masină. Dacă acesta a apăsat butonul atunci facem ,,thePlayer'' și ,,playerCam'' inactive și setăm ,,currenCar'' cu numele mașinii preluat din ,,carName'' . De asemenea sunt activate scripturile ,,CarController'', ,,CarUserControler'', ,,CarAudio'' și CarLights'' pentru a putea folosi masina. Deoarece player-ul a intrat în mașină activam ,,exitTrigger''.
- public void EnterTheCar(): face exact aceeași pași ca funcția ,,Update'', însa fără a verifica dacă ,,triggerCheck'' are valoarea 1. Este nevoie de această funcție pentru a putea avea capacitatea să ,,forțăm'' player-ul în masina fără ca acesta să apese vreun buton.

ExitCar

Acest script este atribuit unui obiect. Cu ajutorul acestuia este posibil să simulăm debarcarea player-ului din mașină.

Scriptul adaugat se numește ,,ExitCar'' și are ca variabile:

- publicGameObject carCam;
- private GameObject playerCam;
- private GameObject thePlayer;
- public GameObject exitTrigger;
- public GameObject theCar;
- public GameObject exitPlace;
- private CurrentCar currentCar;
- public EnterCar enterCar;

Toate variabilele publice sunt setate manual pentru fiecare mașina căruia i se atribuie scriptul.
Funcțiile acestei clase:

- void Start(): Deoarece, în acest stadiu, obiectul ,,Player'' este dezactivat,este nevoie să preluăm referința către acesta din script-ul ,,EnterCar''. Variabilele ,,thePlayer'' și ,,playerCam'' sunt extrase din ,,enterCar'' reprezentând variabilele cu același nume.
- void Update(): Verificăm dacă Player-ul a apăsat butonul care corespunde cu ieșirea din mașina. Dacă acesta a apăsat butonul atunci facem ,,thePlayer'' și ,,playerCam'' active și setăm ,,currenCar'' cu numele ,,None''. De asemenea sunt dezactivate scripturile ,,CarController'', ,,CarUserControler'', ,,CarAudio'' și CarLights'' pentru a putea controla playerul fără a avea vreun efect și asupra mașini. Pentru ca Player-ul sa nu ajungă la locul inițial unde a intrat în masina, setăm poziția acestuia la obiectul ,,ExitCar'',tot langa ușa stângă. Deoarece player-ul a ieșit din mașină dezactivăm ,,exitTrigger''.

4.4.3.4)Dealership

Dealership-ul este locul de unde jucătorul poate achiziționa mașini. Aici se găsesc expuse mașinile disponibile și informațiile necesare despre ele. Fiecare mașina poate fi cumpărată doar dacă jucătorul are level suficient de mare și suficienți bani.

BuyCar

Orice mașina expusă are un script ,,BuyCar'' atașat care are grija ca cerințele sa fie îndeplinite. Acest script este atribuit unui obiect cu funcția de trigger atașat mașinii.
Scriptul adăugat se numește ,,ExitCar'' și are ca variabile:

- public string CarName;
- private CarCollection playerGarage;
- private MoneySystem playerMoney;
- private LevelSystem playerLevel;
- public int pret=0;
- public int levelNeeded=1;
- public int triggerCheck = 0;

Toate variabilele publice sunt setate manual pentru fiecare mașina căruia i se atribuie scriptul.

Funcțiile acestei clase:

- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu triggerul, acesta este considerat un ,,Collider''. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,Player'', atribuim variabilei ,,triggerCheck'' valoarea 1.
- private void OnTriggerExit(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un obiect care a intrat în contact cu triggerul înainte oprește coliziunea acesta este considerat un ,,Collider''. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,Player'', atribuim variabilei ,,triggerCheck'' valoarea 0.
- void Update(): Dacă ,,triggerCheck'' are valoarea 1, înseamna că player-ul este în dreptul ușii și poate intra. Se cauta obiectul cu tag-ul ,,Player'' și se extrag componentele ,,CarCollection'', ,,MoneySystem'' și ,,LevelSystem'' în variabilele ,,playerGarage'' , ,,playerMoney'' și ,,playerLevel''. Următorul pas este să verificăm dacă Player-ul a apăsat butonul care corespunde cu cumpărarea mașinii . Dacă acesta a apăsat butonul atunci verificam sa aibă level-ul(playerLevel) și suma de bani(playerMoney) necesare și de asemenea să nu dețină deja mașina respectivă. Dacă toate condițiile sunt îndeplinite mașina este adaugata la garajul jucătorului(playerGarage).

4.4.3.5)FreshSpawn

Când se face trecerea din scena ,,Garage'' în scenă ,,World'' este nevoie să fie făcut ,,spawn'' mașinii cu care jucătorul a părăsit scena precedenta. Din acest motiv este nevoie sa avem o listă cu toate mașinile și să o alegem pe cea corectă. Acest ,,fresh spawn'' are ca componente:

AllCars

Acest script conține lista cu numele și ,,prefab''-urile tuturor mașinilor pe care player-ul le poate deține.

Scriptul are ca variabile:

- [Serializable]
  public struct Dictionar
  {
  public string name;

```
    public GameObject carPrefab;
    public Dictionar(string Name, GameObject car)
    {
        name = Name;
        carPrefab = car;
    }
}
```

- public List<Dictionar> AllAvailableCars = new List<Dictionar>();

Deoarece în Unity nu exista un timp de date serializabile care să rețină doua tipuri de date diferite, am fost nevoit să creez un tip nou ,,Dictionar''. Acesta reține un nume alături de un ,,prefab'' al mașinii. Pentru a putea reține toate mașinile, avem o variabilă publică ,,AllAvailableCars'' reprezentând o lista cu elemente de tipul creat anterior. Fiecare mașină este adăugată manual aici.

SpawnSelectedCars

Cu ajutorul acestui script facem ,,spawn''mașinii pe care jucătorul a decis să o folosească. Scriptul ,,SpawnSelectedCars" are ca variabile:

- private string carName;
- public GameObject carObject;
- public AllCars cars;
- private GameObject thePlayer;
- private CurrentCar currentPlayerCar;
- public GameObject car;
- public EnterCar enter;
- private int ok = 0;
- private int ok2 = 0;

Toate variabilele publice sunt setate manual pentru fiecare mașină căruia i se atribuie scriptul.

Funcțiile acestei clase:

- void Start(): Se cauta obiectul cu tag-ul ,,Player'' și se extrage componenta ,,CurrenCar''. Deoarece în momentul în care scenă s-a schimbat s-a făcut un autosave, numele mașinii din ,,CurrentCar'' este numele mașinii cu care jucătorul a părăsit garajul. Se cauta în ,,cars'' numele mașinii, ca apoi să fie salvat ,,prefab''-ul acesteia în ,,CarObject''. Dacă a fost găsită mașina, atunci i se face ,,spawn''.
- void Update(): Se caută un obiect cu tag-ul ,,CarTrigger'' și se salvează în ,,car''. Dacă ,,car'' nu este null atunci introducem componenta ,,EnterCar'' în ,,enter''. Din script-ul ,,EnterCar'' folosim funcția ,,EnterTheCar()'' pentru a introduce ,,player''-ul în mașina.

4.4.3.6)RideSystem

După cum se poate regăsi și în titlul jocului ,,Taxi Simulator'' , ideea de baza a jocului este de a putea simula acest job. De aceea este nevoie de un mod de a putea face curse de taxi. ,,RideSystem'' este componenta responsabilă de acest lucru. Aceasta cuprinde mai multe sisteme de stații de taxi formate din :

AiCustomer

Acest script se atașează clienților care urmează să fie transportați. Cu ajutorul acestuia oferim Ai-ului poziția mașinii pentru a se deplasa la ea.
Scriptul ,,AiCustomer'' are ca variabile:

- public Transform target;
- public GameObject Spawn;
- public float TimeToTravel=0;
- public float speed = 4f;
- Rigidbody rig;

Toate variabilele publice sunt setate manual pentru fiecare stație taxi căruia i se atribuie scriptul. Funcțiile acestei clase sunt:

- void Star(): Deoarece acest script se atașează doar obiectelor cu un ,,RigidBody'', putem lua componenta aceasta în variabila ,,rig''. Setăm locul de plecare la ,,Spawn''.
- void FixedUpdate(): Având un target setat, deplasăm caracterul în acel loc.

TaxiCar

Acest script se atașează unui obiect cu funcția de trigger,care este ,,children'' al mașinii jucătorului.Cu ajutorul acestuia vom seta locul unde trebuie să ajungă clientul.

Scriptul ,,TaxiCar'' are ca variabile:

- private TaxiStation taxiStation;
- private int active = 0;

Funcțiile acestei clase sunt:

- void Star(): Verificăm dacă se găsește un obiect cu tag-ul ,,TaxiStation''. În cazul în care s-a găsit selectam componenta ,,TaxiStation'' în variabila ,,taxiStation'' și modifica valoarea variabilei ,,active'' în 1.
- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu triggerul, acesta este considerat un ,,Collider''. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,AiCustomer'' și variabila ,,active'' are valoarea 1, atribuim variabilei ,,clientArrived'' din ,,taxiStation''valoarea 1.
- private void OnTriggerExit(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un obiect, care a intrat în contact cu triggerul înainte, oprește coliziunea acesta este considerat un ,,Collider''. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,AiCustomer'' și variabila ,,active'' are valoarea 0, atribuim variabilei ,,clientArrived'' din ,,taxiStation'' valoarea 0.

TaxiStation

Acesta este, după cum îi spune și numele, stația de taxi, unde se așteaptă clienții. Acest script se ataseaza unui obiect cu funcția de trigger.

Scriptul ,,TaxiStation'' are ca variabile:

- public List<GameObject> aiCustomers = new List<GameObject>();
- public GameObject aiCustomer;
- public AiToCar aiScript;
- public GameObject taxiDestination;
- public TaxiDestination taxiDestinationScript;

- public GameObject clientEntrance;
- public MoneySystem playerMoney;
- public LevelSystem playerLevel;
- public int triggerCheck = 0;
- public int clientArrived = 0;
- public int clientOnBoard = 0;
- public float TimeToArrive = 0;
- public float TimeToGetBack = 0;

Toate variabilele publice sunt setate manual pentru fiecare stație taxi căruia i se atribuie scriptul.

Funcțiile acestei clase:

- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu triggerul, acesta este considerat un Collider. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,PlayerCar'', atribuim variabilei ,,triggerCheck'' valoarea 1.
- private void OnTriggerExit(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un obiect, care a intrat în contact cu triggerul înainte, oprește coliziunea acesta este considerat un Collider. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,PlayerCar'', atribuim variabilei ,,triggerCheck'' valoarea 0.
- void Start():Alegem un număr aleator pentru a selecta un client la întâmplare în variabila ,,aiCustomer'' din lista ,,aiCustomers''. Căutăm un obiect cu tag-ul ,,ClientEntrance'' pentru a-l atribui variabilei ,,clientEntrance''. Găsim obiectul cu tag-ul ,,Player'' și extragem componentele ,,MoneySystem'' și ,,LevelSystem'' în variabilele ,,playerMoney'' și ,,playerLevel''. De asemenea transmitem aceste componente și în ,,taxiDestinationScript'' în variabilele ,,playerMoney'' și ,,playerLevel''.
- void Update():Verificăm ca ,,triggerCheck'' să aibă valoarea 1 (mașina este în stație) și ,,clientOnBoard'' are valoarea 0 (nu avem deja un client în mașina). Odată ajuns în stație, jucătorul apăsa tasta specifica acțiunii prin care declara că este liber și așteaptă un client. Clientul selectat mai sus ,,aiCustomer'' este adus și pleacă din locul de ,,spawn''. Acesta are un timp limita în care poate ajunge la masina, dacă îl depășește înseamna ca ceva este

în neregulă,iar clientul se întoarce la locul de spawn. Dacă clientul a ajuns, atunci acesta intră in masina, iar destinatia acestuia apare pe mapa(taxiDestination.SetActive(true)).

TaxiDestination

Acest script se atașează unui obiect cu funcția de trigger.Cu ajutorul acestuia primim locul unde trebuie sa ducem clientul și finalizam cursa.

Scriptul ,,TaxiDestination'' are ca variabile:

- public GameObject customerDestination;
- private GameObject aiCustomer;
- private AiToCar aiScript;
- private GameObject clientEntrance;
- private MoneySystem playerMoney;
- private LevelSystem playerLevel;
- private TaxiStation taxiStation;
- private int TriggerCheck = 0;
- private int clientOnBoard = 0;
- private float TimeToArrive = 0;

Toate variabilele publice sunt setate manual pentru fiecare destinatie taxi căruia i se atribuie scriptul.

Funcțiile acestei clase:

- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu triggerul, acesta este considerat un Collider. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,PlayerCar'', atribuim variabilei ,,triggerCheck'' valoarea 1.
- private void OnTriggerExit(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un obiect care a intrat în contact cu triggerul înainte oprește coliziunea acesta este considerat un Collider. Verificăm tag-ul atribuit parametrului ,,other'', iar dacă acesta este ,,PlayerCar'', atribuim variabilei ,,triggerCheck'' valoarea 0.

- void Start(): Căutăm un obiect cu tag-ul ,,TaxiStation'' și atribui componenta ,,TaxiStation'' variabilei ,,taxiStation''.
- Void Update(): Atribuim variabilelor ,,clientOnBoard'', ,,aiCustomer'', ,,aiScript'', ,,clientEntrance'', ,,clientOnBoard'' conținutul variabilelor cu același nume din ,,taxistation''. De asemenea căutăm obiectul cu tag-ul ,,PlayerCar'' și reținem ,,expBonus'' și ,,moneyBonus''. Verificăm dacă ,,clientOnBoard''are valoare 1(avem ce client sa aducem) și ,,triggerCheck'' este 1(ne aflăm la destinația data de client). Dacă jucătorul apasă tasta corespunzătoare termin[rii cursei player-ul primește bani și experiența pentru cursa în funcție de distanța parcursă, bonusul dat de mașina și nivel. De asemenea clientul coboară și își continua traseul spre ,,customerDestination''. Dacă după un anumit timp nu a ajuns la destinație, atunci este posibil s[fi apărut o problem[,iar clientul este dezactivat.

4.4.4)Scripturi comune :

4.4.4.1)Teleport

Pentru a face trecerea dintr-o scena în alta este folosit acest șablon.

AutoSave

În fiecare scenă există un ,,Player'' care este setat într-un stadiu default. Pentru a putea păstra progresul dintre scene este nevoie ca la schimbarea scenei să îl salvăm pentru a putea apoi să oferim datele schimbate caracterului din scena următoare.

Scriptul are ca variabile:

- private GameObject player;
- private MoneySystem moneySave;
- private LevelSystem levelSave;
- private LevelSystem expSave;
- private CarCollection garajSave;
- private CurrentCar carNameSave;

Funcțiile acestei clase:

- void Start(): Se caută în scenă un obiect cu tag-ul ,,Player'' și i se atribuie variabilei ,,player''. Se extrag componentele ,,MoneySystem'', ,,LevelSystem'', ,,CarCollection''și

„CurrentCar'' din variabila anterioară și se atribuie variabilelor „moneySave'', „levelSave'' , „expSave'' , „garajSave'' și „carNameSave''.

- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu triggerul, acesta este considerat un „Collider''. Verificăm tag-ul atribuit parametrului „other'', iar dacă acesta este „PlayerCar'' sau „Player'' salvăm datele de mai sus cu ajutorul funcției „SavePlayer'' a scriptului „SaveSystem''

SceneExit

În momentul în care jucătorul decide să părăsească garajul este nevoie de un script care simulează ieșirea pe ușă a caracterului și schimbă scena activă. Acest script este atașat unui obiect cu funcția de trigger.

Scriptul are ca variabile:

- public string sceneToLoad;

Variabila publică este setată manual, deoarece fiecare teleport duce la o scenă diferită.

Funcțiile acestei clase:

- private void OnTriggerEnter(Collider other): Aceasta este o funcție specifică unui trigger. În momentul în care un alt obiect a intrat în contact cu triggerul, acesta este considerat un „Collider''. Verificăm tag-ul atribuit parametrului „other'', iar dacă acesta este „PlayerCar'' sau „Player'' schimbăm scena cu ajutorul funcției „LoadScene'' din SceneManager.

4.4.4.2)Save System

Pentru a salva progresul caracterului este folosit acest șablon.

SaveData

Pentru a putea salva datele în mod binar este nevoie ca ele să fie serializabile. Aceasta este o clasa ajutătoare pentru a transforma elementele care trebuie să fie salvate.

Scriptul are ca variabile:

- public int moneySave;

- public int levelSave;
- public float expSave;
- public List<string> garajSave = new List<string>();
- public string currentCar;

Funcțiile acestei clase:

- public SaveData(int money,int level, float exp, List<string>garaj, string carName): Aceasta este o funcție care primește ca parametri elementele player-ului care trebuie sa fie salvate și le salvează în variabilele proprii.

SaveSystem

Acesta este scriptul care se ocupă efectiv cu a încarca și a prelua progresului player-ului.

Funcțiile acestei clase:

- public static void SavePlayer(int money, int level, float exp, List<string>garaj, string currentCar): Setam un ,,formatter'' de tip BinaryFormatter, un path pentru locul unde vor fi reținute datele și un ,,stream'' de tip ,,FileStream'' pentru a accesa fișierul creat anterior. Într-o variabila locală de tipul ,,SaveData'' salvăm elementele transmise ca parametrii. Apoi cu ajutorul ,,formatter''-ului salvăm datele în mod binar în fișier.
- public static SaveData LoadPlayer (): Într-o variabila locală ,,path'' de tipul string oferim locul fișierului unde am făcut anterior salvarea.Dacă fișierul există , cu ajutorul unui ,,formatter'' de tip BinaryFormatter și a unui ,,stream'' de tip ,,FileStream'' extragem informațiile într-o variabila tipul ,,SaveData''.Dacă fișierul nu există atunci returnăm null.

SaveNLoad

Acest script se atașează ,,Player''-ului din fiecare scenă.

Funcțiile acestei clase:

- private void Awake():Aceasta funcție se apelează o singură dată, la început, și este responsabilă pentru a oferi datele despre progres ,,Player''-ului din scena curentă. Acest lucru este posibil prin apelarea funcției ,,LoadPlayer'' declarată mai jos.
- public void SavePlayer():Se caută un obiect cu tag-ul ,,Player'' și i se atribuie variabilei locale ,,player''. Se extrag componentele ,,MoneySystem'', ,,LevelSystem'', ,,CarCollection'' și ,,CurrentCar'' și se se salvează cu ajutorul funcției ,,SavePlayer'' din script-ul ,,SaveSystem''.

- public void LoadPlayer(): Într-o variabilă de tipul ,,SaveData'' se salvează rezultatul funcției ,,LoadPlayer''. Se caută un obiect cu tag-ul ,,Player'' și i se atribuie variabilei locale ,,player''. Se extrag componentele ,,MoneySystem'', ,,LevelSystem'', ,,CarCollection'' și ,,CurrentCar'' și li se atribuie valoarea salvată în sistem.

3.Propuneri de dezvoltare ulterioară

Pentru a îmbunătăți experiența de joc există mereu lucruri care pot fi adăugate sau schimbate la un joc. Uneori cu ajutorul feedback-urilor de la persoanele care folosesc aplicația găsim idei care să îmbunătățească jocul, alteori echipa de dezvoltatori devine propriul critic și decide ce schimbări pot fi făcute. În opinia mea, jocul ,,Taxi Simulator'' ar deveni și mai bun cu următoarele idei:

- Meniului ,,Option'' să i se adauge și controlul asupra calității jocului. Acest lucru este foarte folositor, deoarece, atât în stadiul acesta ,cât și în viitor este posibil ca experiența de joc să fie stricată de incapacitatea anumitor calculatoare personale de a face față cerințelor la rezolutie maxima.
- Să existe și penalități din banii cursei. Deși sistemul prin care se calculează valoarea unei curse nu este unul rudimentar, acesta luând în calcul distanța, nivelul jucătorului, cât și bonusurile oferite de mașina condusă,trebuie să existe și penalități pentru a încuraja un comportament asemănător cu cel din viața reală.
- Să existe un sistem de review. Odată ce sistemul de penalițăti este implementat, următorul sistem ce ar veni ca o completare este cel de review. Asemanatoru cu sistemele din viața reală, în funcție de performanțele jucătorului, acesta primește un punctaj la sfârșitul fiecărei curse. Acest lucru ar servi ca un alt mijloc de menținere a interesului din partea jucătorilor.

- Animații pentru îmbarcarea și debarcarea din autovehicul. Deși sistemul actual nu este un impediment în desfășurarea jocului, setarea unor animații ar ajuta la un sentiment mai puternic de simulare a vieții reale per total.
- Un garaj auto unde se pot tuna și personaliza mașinile. În cele mai multe cazuri,persoanele care joacă jocurile cu automobile au o oarecare pasiune pentru acestea. Prin posibilitatea de a personaliza mașina se leagă o anumită legătură, acesta fiind un alt factor important care ajută la longevitatea jocului prin menținerea interesului jucătorului.
- Posibilitatea de a alege ordinea mașinilor în garaj.
- Posibilitatea de a cumpăra alte case si garaje. Deși nu sunt o parte crucială a jocului, majoritatea jucătorilor își doresc sa dețină numeroase proprietăți virtuale. De asemenea, odată cu dezvoltarea jocului, și apariția unor noi mașini, spațiul unui singur garaj nu va mai fi suficient, posibilitate de cumpărare a altor garaje fiind o necesitate.
- Alimentarea mașinii. Pentru o complexitate a jocului ridicată și o reprezentare autentică, este nevoie ca mașina să trebuiască sa fie alimentată. Acest lucru reprezintă un alt lucru pe care jucătorul trebuie să-l ia în calcul să își folosească banii, oferind astfel întrebuințări multiple valutei.
- Ciclu noapte-zi. Pentru a simți un progres este nevoie de a simți trecerea timpului. Un ciclu noapte-zi ar putea deschide și alte oportunități de gameplay și ar ajuta și la întreruperea rutinei jucătorului.
- Mai multe stiluri de taxi. Pe lângă tradiționalul mod de a practica această meserie, în zilele noastre au apărut și opțiuni mai moderne. S-ar putea introduce și o aplicație asemănătoare cu ,,Uber'' sau ,,Bolt'', fiecare aplicație venind cu avantajele și dezavantajele sale.
- Apariția unor misiuni secundare. Pentru a încuraja jucătorul să viziteze întregul oraș, se pot plasa diferite obiecte de colectat în locuri mai puțin vizibile.Acest lucru oferă jucătorului un nou scop care îi poate trezi interesul.
- Apariția unui ,,leaderboard'' global. Unul dintre cei mai mari factori care țin un joc ,în ,,viața'' este simțul de competiție al oamenilor. Apariția unui clasament ar putea ambiționa multe persoane să ajungă în top.
- Adăugarea de noi caractere și personalizarea acestora. Acest lucru ajută în continuare cu implicarea emoțională a jucătorului și de asemena cu creșterea în importanță a banilor.

4.Concluzii

Luând în considerare toate cele prezentate mai sus despre industria jocurilor, în speța a celor simulator, care în ultimii ani au evoluat foarte mult, consider că tema aleasă este una de actualitate și cu o viziune pe termen lung. Această opinie este susținută și de prognoza pieței globale a simulatoarelor de jocuri care este de 13.378 milioane de dolari până în 2026, în creștere de la 4.320 milioane de dolari în 2018, la o rată de 17,2%.

Rezultatele obținute în cadrul proiectului ne ilustrează că ne putem apropia de un joc de calitate, care poate forma o comunitate în jurul acestuia, chiar și cu resurse limitate. Am ajuns într-un punct în care informațiile disponibile și evolutia tehnologica permit multor creatori să scoată un produs bun. Faptul că din ce în ce mai multă tehnologie, necesară dezvoltării unui joc,este oferită gratuit,atrage un număr din ce în ce mai larg de persoane. Aceasta este o veste bună pentru industrie, deoarece creșterea numărului de persoane rezulta într-o dezvoltare direct proporțională a acestui sector.

În urma functionalitaților prezentate anterior consider ca proiectul prezentat este un demo complet care arată un început promițător al unui joc de succes. Totodată, putem concluziona că acest proiect poate fi adaptată și utilizată în alcătuirea unui nou timp tip de joc care implica componenta mașinilor sau care necesita un loc de desfășurare al acțiunii. Acest lucru deschizând alte viitoare oportunități și idei de joc.

Nu în ultimul rând, procesul a condus către aprofundarea cunoștințelor din cele două

domenii studiate: modelarea 3d și folosirea unui game engine. Am descoperit noi tehnologii care au ajutat la bazele jocului și au micșorat distanța între echipele mari și dezvoltatorii singuri, oferind noi orizonturi în acest domeniu.

5.Bibliografie

[1] Josh Petty, *What is Houdini & What Does It Do?,*
https://conceptartempire.com/what-is-houdini-software/

[2] *Video Game Industry*, https://en.wikipedia.org/wiki/Video_game_industry

[3] *Video Game*, https://en.wikipedia.org/wiki/Video_game

[4] J.Clement, *Video game industry - Statistics & Facts,*
https://www.statista.com/topics/868/video-games/

[5] Dwight Pavlovic, *Video Game Genres: Everything You Need to Know,*
https://www.hp.com/us-en/shop/tech-takes/video-game-genres

[6] Phil Owen, *What Is A Video Game? A Short Explainer,*
https://www.thewrap.com/what-is-a-video-game-a-short-explainer/

[7] https://www.researchdive.com/179/gaming-simulator-market

[8] Josh Petty, *What is Houdini & What Does It Do?,*
https://conceptartempire.com/what-is-houdini-software/

[9] *User Manual*, https://www.sidefx.com/products/houdini/fx-features/

[10] *Houdini Software*, https://en.wikipedia.org/wiki/Houdini_(software)

[11] Rachel Kim, *WHY IS HOUDINI THE FUTURE OF 3D AND VFX?,* https://infocusfilmschool.com/houdini-future-3d-vfx/

[12] *Unity Game Engine,* https://en.wikipedia.org/wiki/Unity_(game_engine)

[13] Lindsay Schardon, *Making your Dream Games: What is Unity?*, https://gamedevacademy.org/what-is-unity/

[14] *Unity Asset Store,* https://assetstore.unity.com/

[15] *Gaming Simulator Market Size, Share & Trends Analysis Report,* https://www.grandviewresearch.com/industry-analysis/gaming-simulator-market

[16] *Third Person Controller,* https://assetstore.unity.com/packages/tools/game-toolkits/third-person-controller-basic-locomotion-free-82048

[17] *Unity Standard Assets,* https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351

[18] *Unity User Manual*, https://docs.unity3d.com/Manual/index.html