UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Kingdom HazARd

propusă de

Răzvan-Adrian Cimpoeșu

Sesiunea: iulie, 2018

Coordonator științific

Drd. Colab. Florin Olariu

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Kingdom HazARd

Răzvan-Adrian Cimpoeșu

Sesiunea: iulie, 2018

Coordonator științific

Drd. Colab. Florin Olariu

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemntatul(a) ……
domiciliul în ……..
născut(ă) la data de ……..…., identificat prin CNP ……..……..., absolvent(a) al(a) Universității „Alexandru Ioan Cuza" din Iași, Facultatea de ……. specializarea …… promoția ……., declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul: _____
_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „ Kingdom HazARd”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, dată

Absolvent Răzvan-Adrian Cimpoeșu

_____

(semnătura în original)

Cuprins

Introducere

Abstract

Lucrarea de licență are ca scop aplicarea modelului realității augmentate asupra unui gen clasic de joc în care jucătorul va trebui să se dea dovadă de o gândire rapidă si de capacitatea de a forma o strategie de moment pentru a se confrunta cu o inteligență artificială .

În momentul de față, Unity este cel mai căutat motor de dezvoltare a jocurilor video, arătând o creștere enormă atât în popularitate cât si în posibilitățile oferite dezvoltatorului, printre aceste posibilități, regăsindu-se si SDK-ul Vuforia, ce facilitează interacțiunea cu mediul realității augmentate.

Piața limitată în acest domeniu împreună cu mediul de dezvoltare propice care este pus la îndemâna dezvoltatorilor fac abordarea unui proiect ce se folosește de realitate augmentată o alegere îndrăzneață dar totodată foarte justificată și accesibilă.

Motivație

Încă din copilărie am fost captivat de lumea jocurilor pe calculator. Ceea ce pe atunci era o modalitate de petrecere a timpului liber, a devenit cu timpul o modalitate de interacțiune cu prietenii care îmi împărtășeau acest hobby, mai târziu devenind o pasiune. Spun pasiune întru-cât, de la o anumită vârstă, circa 14-15 ani, am început să îmi pun întrebări precum: „Oare care este logica din spatele acestui joc? ”, „Cum anume este posibil să pot mișca un caracter liber pe o hartă? ”, „ Oare pot face și eu așa ceva? ”.

Deși în acel punct nu aveam interacțiune cu lumea programării, răspunsurile au început să apară și m-au motivat să vreau să aflu cât mai multe, până în punctul în care am deschis prima dată Unity și am început să îmi fac propriul joc. Bineînțeles aspirațiile mele din acel moment erau enorme și visam să fac un joc care mă va face milionar peste noapte, un joc care va captiva pe toată lumea și care îmi va aduce faima la nivel mondial.

Reușisem să fac o imitație de Mario în care jucătorul putea să miște un caracter ce avea ca scop să se ferească de obstacole și să colecteze diferite obiecte pentru a putea termina jocul. Deși produsul final era o copie clară a unui joc celebru, cu o implementare mult mai slabă, în acel moment pentru mine a fost o mare realizare, reușisem să îmi răspund la o parte din întrebările pe care le aveam vizavi de implementarea jocurilor și de asemenea reușisem să creez un joc pe care îl puteam arăta prietenilor și să pot spune: „ Eu am făcut asta! ”.

După această experiență, interacțiunea mea cu jocurile video a fost strict una de consumator, doar că de această dată având un minim de cunoștințe în domeniu, fapt care îmi permitea să am o oarecare înțelegere asupra implementării jocului care se desfășura pe ecranul calculatorului meu. Gândul de a mai da o șansa dezvoltării de jocuri mi-a tot trecut prin cap în acea perioadă însă nu am putut veni cu o idee originală care, cel puțin în opinia mea să prindă la public așa că toate planurile mele au rămas în acel punct la statutul de idee.

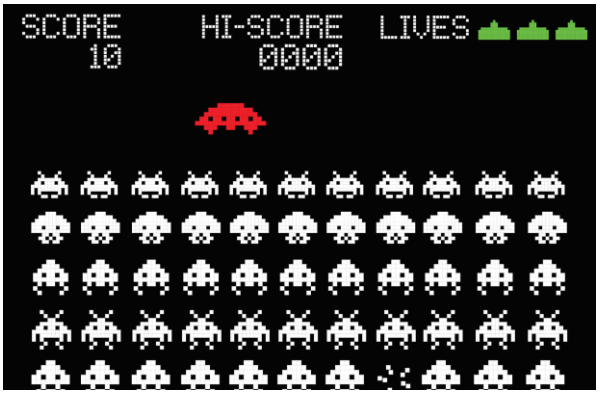
Între timp un nou trend pe piața dezvoltatorilor de jocuri a apărut la orizont, cel al Realității augmentate, în care jucătorul putea aduce elemente din jocuri video în lumea reală, aspect care mi s-a părut revoluționar și care deschide o mulțime de noi posibile abordări asupra jocurilor consacrate. Din acest motiv am început să caut titluri care să se încadreze în acest gen, fiind deschis la mai orice tip de joc, ideea principală fiind explorarea domeniului. Am fost dezamăgit de opțiunile care mi-au fost puse la dispoziție de piața jocurilor video, fapt ce m-a motivat să îmi lărgesc aria de înțelegere a dezvoltării jocurilor spre acest nou domeniu, pentru a căpăta abilitatea de a integra acele idei pe care le aveam în trecut în această tehnologie a prezentului.

Context

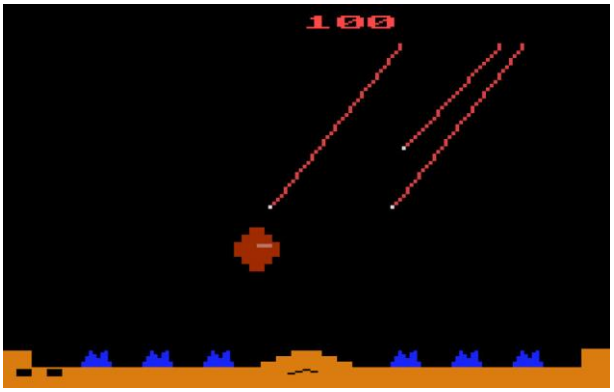
Conform unui studiu realizat de către revista *”The Atlantic”* în anul 2011, vârsta medie a unui jucător de jocuri video este de 34 de ani. În plus, graficele indică faptul că 60% dintre consumatori sunt de genul masculin. Potrivit unei statistici realizate de către Entertainment Software Association, în urma datelor oferite de către NPD Group - o companie de studiu de piață, industria jocurilor video a vândut 273 de milioane de produse în anul 2009, ajungând la un profit surprinzător de 10.5 miliarde de dolari.

Jocurile de Tip Tower Defense(TD) sunt un sub gen al jocurilor de strategie unde scopul este apărarea teritoriului și/sau proprietăților prin blocarea strategică a căii atacatorilor folosind diferite obstacole. În general acest lucru înseamnă construirea de structuri defensive, care au ca

scop împiedicarea avansării, blocarea sau chiar eliminarea inamicilor. Acest gen de joc își are originea în așa-zisa „era de aur a jocurilor video” (1980-2000) când titluri precum „Space Invaders”(1978) (Figura 1) sau „Missle Command”(1980) (Figura 2) puneau la dispoziție jucătorilor o interfață simplistă, un set de inamici cu comportament previzibil și câteva mecanisme de bază. În titlurile mai sus menționate jucătorul deținea controlul asupra unei navete spațiale sau a unui centru de comandă, situat în partea de jos a ecranului iar scopul acestuia era să împiedice proiectilele inamicilor aflați de cealaltă parte a ecranului să atingă unitatea controlată.



(Figura 1) Space Invaders.



(Figura 2) Missle Command.

Elemente de bază într-un joc de tip Tower Defense

-*teritorii* sau *posesii* (colectiv numite „**bază**”) care trebuie aparate de către jucător;

-baza trebuie să supraviețuiască atacurilor mai multor **valuri de inamici** , care pe parcurs ce jocul progresează vor deveni mai puternici și/sau mai numeroși;

-plasarea de elemente defensive precum **ziduri** și/sau **turnuri** în calea valurilor de inamici pentru a apară baza.

Ceea ce diferențiază jocurile de tip TD de alte jocuri de strategie precum titlul mai sus menționat „Space Invaders” este posibilitatea jucătorului la o amplasare strategică a mecanismelor defensive. În general dar nu întotdeauna jucătorul are o perspectivă obiectivă asupra acțiunii care se petrece, obiectul jocului fiind baza și deciziile pe care jucătorul le ia.


(Figura 3)”Rampart” dezvoltat de "Atari Games".

Conceptul de Tower Defense propriu zis este relativ tânăr comparativ cu alte genuri din lumea jocurilor, acestea găsindu-și începutul în anul 1990 cu un joc de tip arcade dezvoltat de „Atari Games” numit „Rampart” (Figura 3). Succesul masiv de care a beneficiat acest titlu a dus în perioada următoare la o lansare a lui pe o varietate de platforme cum ar fi „Game Boy”, „Super NES”, „Playstation”, „Xbox” și multe altele. Acest succes a luat cu asalt lumea jocurilor-video, fapt ce a dus la crearea de hărți pentru jocuri populare, care au preluat modelul Tower Defense, printre care „Starcraft”, „Age of Empires II” și „Warcraft III”.

Datorită posibilităților de a crea hărți, comunitatea de jucători a ridicat un val de astfel de creații, fapt care eventual a dus la adoptarea genului în mod oficial de cate „Warcraft III”. După cum menționează Phillipa Avery în lucrarea sa „*Computational Intelligence and Tower Defense Games*” : „Una din ce cele mai populare versiuni ale acestor modele de harți Tower Defense, și nediscutabil primul joc cu adevărat originar acestui stil a fost harta Tower Defense pentru expansiunea de „Warcraft III : The Frozen Throne”.(Figura 4). Această adiție a semnalat intrarea genului Tower Defense pe piață și a stârnit un interes asupra genului la un nivel înalt.

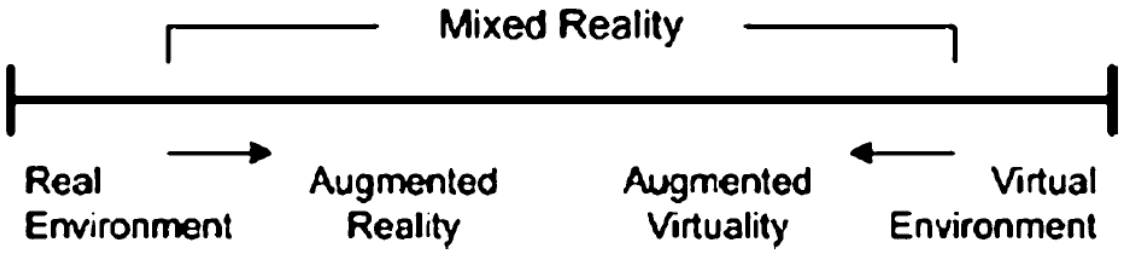
(Figura 4) Warcraft III : The Frozen Throne , modul de Tower defense.

Realitate virtuala/Realitate Augmentată

Definim Realitatea Augmentata (AR) ca o modalitate interactivă de a percepe lucrurile din lumea reală într-un mod „augmentat”, îmbunătățit, acest lucru realizând-se adăugând elemente virtuale, generate de calculator. Continuumul „Realitate – Virtualitate” a lui Milgram este definit de Paul Milgram și de Fumio Kishino ca un pendul ce oscilează continuu între Realitate augmentată

și Virtualitate augmentată. Realitatea augmentată fiind totuși mai aproape de realitatea pe când Virtualitatea augmentată tinde la un mediu pur virtual care înlocuiește cu totul lumea reală, după cum este exemplificat în Figura 5

Scopul Realității augmentate este să simplifice viața utilizatorului nu doar aducând informații virtuale în apropierea sa imediată, ci și oferind o imersivitate într-o experiență precum ar fi un joc video sau precum ar fi privirea unui film. Pe când Realitatea virtuala , sau mediul virtual, așa cum îl numește Milgram, acaparează utilizatorul într-u-totul în lumea 100% sintetică, generată pe calculator fără a mai avea vreun contact cu lumea reală.



(Figura 5) Continuumul „Realitate – Virtualitate ”a lui Milgram

Contribuții

Am reușit să creez un joc video care se încadrează într-un gen clasic dar care folosește o abordare cu totul diferită, cea a realității augmentate, permițând jucătorului să aducă acțiunea desfășurată în mod tradițional pe ecranul unui calculator, într-o lume complet virtuală, pe masa din sufragerie.

În partea de abordare generală a jocului am reușit să adaug elementul de puzzle în care jucătorul poate alege o abordare diferită de la joc la joc, având posibilitatea de a ghida inamicii pe hartă în modul dorit, plasarea obstacolelor realizând-se pe baza unei grile care definește zone unde acestea pot fi amplasate.

O altă contribuție este perspectiva pe care o are aplicația dezvoltată de mine, în timp ce alte jocuri de tip TD au o perspectivă fixă, în aplicația Kingdom HazARd jucătorul poate privi acțiunea prin ochii unui inamic, fie de pe un vârf de munte, ba chiar din interiorul unei căsuțe.

Am reușit să creez o aplicație care se diferențiază de alte titluri ale aceluiași gen aducând elemente originale si folosindu-mă de o implementare proprie, ghidată de creativitate.

Capitolul I. Arhitectură și implementare

I.1 Scurtă descriere a aplicației

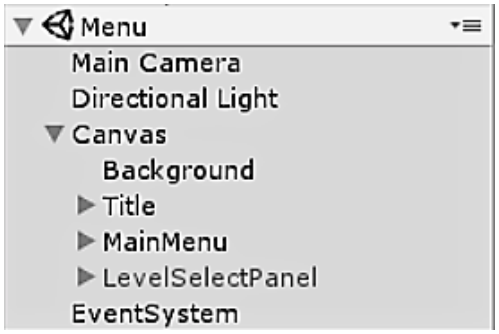
Aplicația Kingdom HazARd este un joc de strategie tip TD dezvoltat folosind motorul de dezvoltare de jocuri Unity, în care partea de scripting și programare au fost realizate folosind limbajul de programare C#. Pentru aducerea aplicației in lumea reală m-am folosit de un SDK (Software developer kit) pe care Unity îl pune la dispoziție și anume Vuforia.

Abordarea jocului este una noua, atât pentru genul său de jocuri cât si pentru jocurile aparținând de AR . Față de titluri clasice ale genului, menționate mai sus, aplicația Kingdom HazARd introduce elemente de puzzle și complexitate, introduce o perspectivă complet liberă asupra modului în care jucătorul poate juca acest joc.

Jocul dorește să forțeze persoana care stă în spatele telefonului să gândească rapid și să se miște în același timp, poziționarea față de suprafața de joc fiind un aspect important, întru-cât unele zone pot fi vizibile doar din unghiul potrivit.

I.2 Arhitectura aplicației

Arhitectura aplicației este dată de felul în care obiectele din joc sunt organizate în interiorul unei scene. Aplicația este construită pe 3 astfel de scene, una dintre ele fiind implementarea meniului principal iar celelalte 2 fiind asociate fiecărui nivel implementat.



(Figura 6) Arhitectura meniului

În scena destinată meniului este prezent un obiect canvas (tip de obiect in Unity care pune la dispoziție o suită de obiecte 2D destinate implementării de interfață) pe care sunt expuse elemente precum butoane și panouri, acestea fiind grupate conform Figurii 6.

În scenele corespunzătoare celor 2 harți obiectul în jurul căruia se centrează întreaga arhitectură este "Imagetarget", acesta fiind obiectul de legătură dintre mediul virtual generat în Unity și lumea reală. Obiectul Imagetarget este corespondentul unei imagini existente în lumea reală care atunci când va fi detectată de către aplicație, toate obiectele care îi sunt copii în ierarhia din Figura7 vor apărea în scenă și analog la pierderea referinței către obiect de către camera dispozitivului de unde va fi rulată aplicația, toate acele obiecte vor fi dezactivate.

În interiorul scenei există 2 obiecte de tip canvas asemănătoare cu cele întâlnite la meniu. După cum se poate observa unul din aceste obiecte se află înafara ierarhiei obiectului Imagetarget iar altul în interior.

(Figura 7) Arhitectura unui nivel

Obiectul din exterior, „AlwaysOnCanvas" gestionează interfața care va fi vizibilă utilizatorului pe toată durata de viață a scenei , pe când obiectul „InGameCanvas" gestionează mecanismele care au legătura directă cu jocul și care nu ar putea fi active fără ca imaginea țintă să fie detectată, și jocul încărcat în scenă.

I.3 Implementarea aplicației

I.3.1 Clasa monoBehaviour

Clasa MonoBehaviour este clasa de bază din care toate clasele derivă în cadrul unui script în Unity. Pentru ca o instanță a clasei să fie creată aceasta trebuie atribuită unui game-object care să fie pus în scenă și să fie activ. Unity va rula toate scripturile care moștenesc această clasă pe un singur fir de execuție cu o viteza egală, acest aspect reprezentând numărul de cadre redate pe secundă (mai târziu in document fiind referențiat ca “frame-rate”).

Odată cu moștenirea clasei MonoBehaviour vom avea acces la o serie de metode implicite care sunt esența programării în Unity, aceste metode auto apelând-se în anumite momente bine definite. Aceste metode vor fi populate cu scriptul corespunzător logicii dorite, pe lângă aceste metode putând fi definite altele care să servească necesitatea dezvoltatorului aplicației. Printre cele mai importante regăsim:

Metoda Awake (Script 0.1) care se apelează o singură dată la încărcarea primului obiect care are atașat scriptul în momentul rulării aplicației. Această metodă poate fi considerată un echivalent la constructor întru-cât aceasta se va apela o singură dată per instanță de script, fiind recomandat ca în metoda Awake să fie instanțiate variabile având în vedere faptul că aceasta este apelată o singură dată pe întreaga durată de viată a scriptului.

```
private void Awake()
```

(Script 0.1) Reprezentarea in cod a metodei Awake

Metoda Start (Script 0.2) este apelată imediat după metoda Awake tot la instanțierea obiectului cărui ii este asociat scriptul însă spre deosebire de metoda descrisă mai sus, în această metodă se poate realiza inițializarea proprietăților ce țin de instanța curenta de obiect, ne fiind legată de celelalte obiecte care conțin scriptul. Asemănător cu metoda Awake, metoda Start() se apelează o singura data pe durata de viată a obiectului.

```
private void Start()
```

(Script 0.2) Reprezentarea in cod a metodei Start

Metoda Update (Script 0.3) este una dintre cele mai importante funcționalități oferite de clasa MonoBehaviour întru-cât, prin intermediul ei este dictat comportamentul obiectului care are asociat scriptul. Această metodă se va auto apela de 60 de ori pe secundă, fapt care dictează frame-rate-ul aplicației, excepție făcând cadrele în care obiectul nu este activ. Spre deosebire de metodele Awake și Start care se apelează o singură dată, apelarea continuă a metodei împreună cu implementarea în interiorul ei a unei logici corespunzătoare, asigură obiectului în cauză o funcționalitate fluenta.

```
private void Update()
```

(Script 0.3) Reprezentarea in cod a metodei Update

O altă metodă importantă în construcția aplicației Kingdom HazARd este metoda SartCourutine (Script 0.4) folosită în logica de generarea a valurilor de inamici descrisă mai jos. Execuția unei corutine poate fi oprită în orice punct folosind cuvântul cheie „yield”. Valoare care este returnată folosind „yield” menționează în cat timp execuția va fi reluată. Pentru a putea apela o metodă folosind SartCourutine, această metodă trebuie să fie de tip IEnumerator și să conțină un return de tip „yield”.(Script 0.4)

```
StartCourutine(SpawnWave());
...
IEnumerator SpawnWave(){
...
      yield return new WaitForSeconds(0.5f);
...
}
```

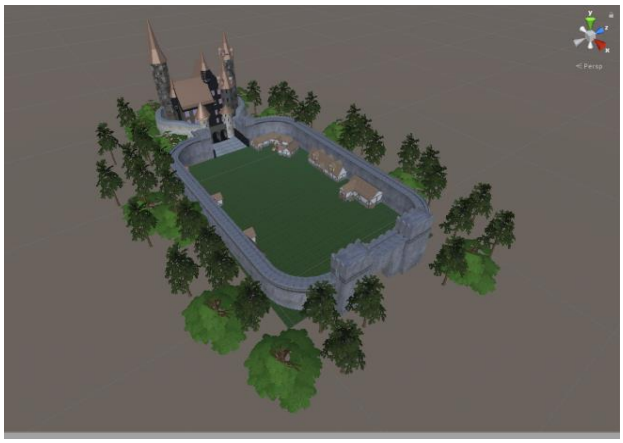
(Script 0.4) Reprezentarea in cod a metodei StartCourutine si tipul de funcție care trebuie apelată

I.3.2 Entități ale aplicației

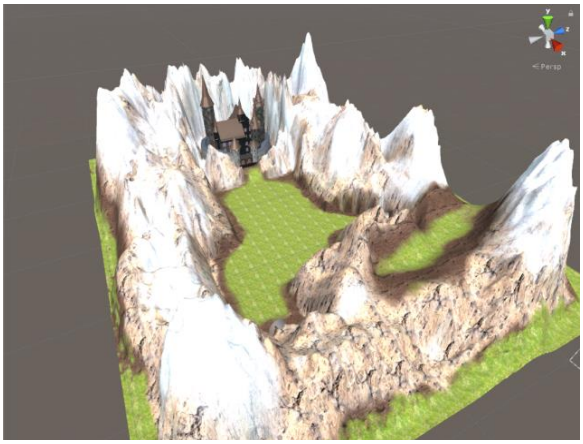
Harta este zona unde toate elementele din joc interacționează între ele și unde întreaga acțiune se va desfășura. Aici se vor genera inamicii care trebuie să ajungă la punctul de destinație (baza) și tot aici prin intermediul matricei de construcții (ce urmează a fi explicată in acest subcapitol) jucătorul va trebui să blocheze calea inamicilor folosind structurile defensive (de asemenea explicate in continuare in acest subcapitol). Aplicația pune la dispoziție două hărți predefinite în care sunt ilustrate două procedee diferite de asamblare a unei scene.

În prima hartă(Figura 8) este prezentat un mediu rural cu tentă medievală care este generat folosind o suită de obiecte 3D cum ar fi: ziduri, case, copaci e.t.c, care au fost ajustate în mărime, rotație și amplasare pentru a crea un peisaj ce oferă un sentiment de veridicitate.

A doua hartă(Figura 9) prezintă o altă modalitate de construire a mediului, folosind o unealtă pusă la dispoziție de Unity, aceasta fiind terenul. Generarea terenului a presupus adăugarea unui game object de tip “terrain” pe care l-am modificat folosind procedee precum ajustarea înălțimii, netezirea marginilor, colorarea lor, e.t.c.



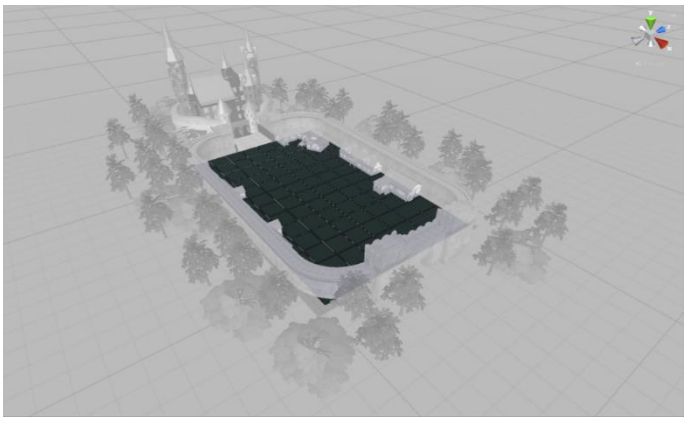
(Figura 8) Harta corespunzătoare primului nivel



(Figura 9) Harta corespunzătoare celui de al doilea nivel

Matricea de construcții este elementul care face posibilă localizarea unor zone de construcție definite pentru jucător. Matricea este alcătuită din 2 straturi suprapuse, unul fiind vizibil jucătorului, care deservește ca zonă de interacțiune pentru procedeul de generare de construcții defensive (explicat în secțiunea următoare a acestui subcapitol), iar celălalt fiind folosit pentru generarea efectivă a elementelor defensive. Un strat este alcătuit dintr-un număr predefinit de zone pătrate denumite noduri, un nod fiind spațiul alocat unui singur element defensiv la un moment dat.

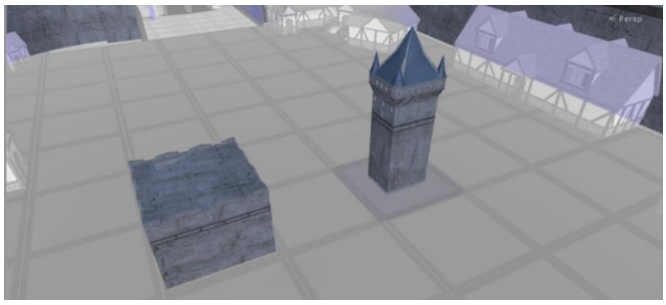
În stratul destinat interacționării cu jucătorul, nodul va juca rolul obiectului care va fi identificat de RaycastHit (explicat în capitolul I.3.2), urmând ca structura care urmează a fi construită să se genereze pe nodul corespunzător din stratul destinat generării obiectelor. Logica din spate presupune identificarea nodului din primul strat, identificarea nodului corespunzător din stratul de generare, verificarea dacă acel obiect are deja un obiect asignat și de asemenea verificarea dacă prețul acelei structuri este mai mic decât bugetul jucătorului în acel moment șimai apoi generarea sau nu a structurii defensive.



(Figura 10) Matricea de construcții

Această abordare pe 2 straturi a fost necesară încât în momentul în care se efectuează generarea unui game object, acesta se va genera ca și copil (d.p.d.v ierarhic) al obiectului referențiat iar stratul care este destinat interacțiunii cu utilizatorul poate deveni inactiv pe parcursul execuției(vezi II.2.2 pentru un astfel de caz), în acel caz făcând inactive și structurile defensive aferente.

Elementele Defensive în aplicația Kingdom HazARd sunt reprezentate de ziduri și turnuri (Figura 11). Zidul este un element care are rolul de a bloca întreaga suprafață a nodului în care este plasat, obstrucționând astfel calea inamicilor pe suprafața acelui nod. Zidurile nu vor riposta în niciun fel față de inamici, unicul lor rol fiind acela de a dirija deplasarea acestora.



(Figura 11) Structuri defensive (stânga: zid , dreapta: turn)

Turnurile sunt atât un element defensiv cât și ofensiv. În momentul în care jucătorul va plasa un turn pe matricea de construcții, acesta va ocupa ca și spațiu doar centrul nodului respectiv, inamicii fiind în continuare capabili să treacă pe suprafața nodului unde a fost amplasat turnul, dar ocolind corpul acestuia. Spre deosebire de ziduri, turnurile au rolul nu doar de a bloca parțial mișcarea inamicilor ci și de a ataca. În momentul amplasării un turn va face o verificare în care va identifica toți inamicii aflați în raza sa (Script 1) (definim raza turnului ca o zonă circulară ce are ca punct central în sine).

```
void UpdateTarget()
    {
        GameObject[] enemies = GameObject.FindGameObjectsWithTag(enemyTag);
        float shortestDistance = Mathf.Infinity;
        GameObject nearestEnemy = null;

        foreach(GameObject enemy in enemies)
        {
            float distanceToEnemy = Vector3.Distance(transform.position,
                                    enemy.transform.position);
            if(distanceToEnemy < shortestDistance)
            {
                shortestDistance = distanceToEnemy;
                nearestEnemy = enemy;
            }
        }

        if (nearestEnemy != null && shortestDistance <= range)
        {
            target = nearestEnemy.transform;
        }

    }
```

(Script 1) Script destinat detectării inamicilor dintr-o anumita rază și selectarea unei instanțe de inamic

Odată ce turnul și-a stabilit o țintă, acesta va trage (Script 2) la un interval de timp predefinit inițial dar care poate fi micșorat pe parcursul jocului. Mecanismul de tragere constă în instanțierea unui game object de tip “Bullet”, ce are atașat un script care ii permite să urmărească o țintă dată și de asemenea definește puterea acelui glonț (cantitatea de viață care va fi scăzută din inamicul lovit).

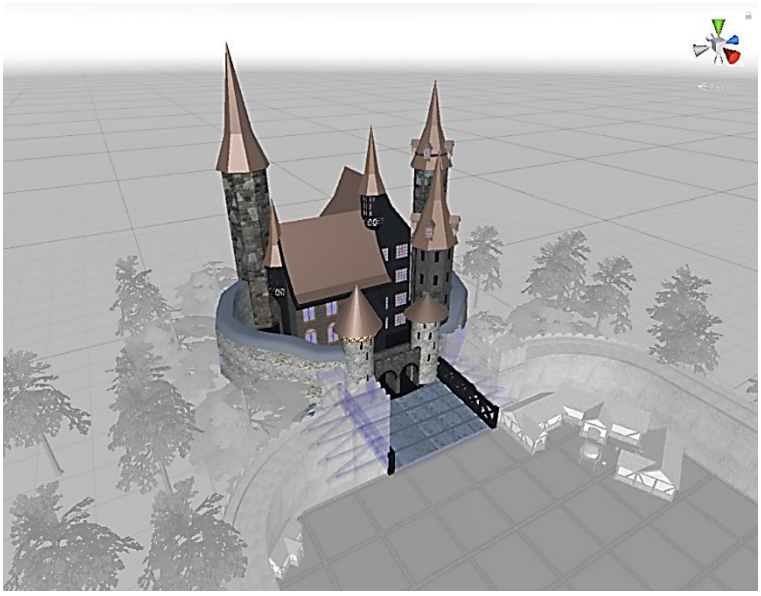
(Script 2) Script ce determină comportamentul obiectului de tip glonț

```
void Shoot()
    {
        GameObject bulletGO = Instantiate(bulletPrefab, firePoint.position,
                              firePoint.rotation);
        Bullet bullet = bulletGO.GetComponent<Bullet>();
        bullet.power = power;

        if (bullet != null)
        {
            bullet.Seek(target);
        }}
```

Caracteristici ale turnurilor precum micșorarea intervalului în care acesta va trage următorul proiectil, raza de detecție a inamicilor și puterea unui glonț pot fi toate îmbunătățite folosind sistemul de îmbunătățiri din joc, în care jucatul va putea cumpăra aceste îmbunătățiri cu banii obținuți din eliminarea inamicilor.

Baza, în aplicație fiind reprezentată de un model 3D al unui castel (Figura 12), este elementul de la capătul opus al harții față de punctul de generare a inamicilor, care trebuie apărat folosind în mod strategic amplasarea elementelor defensive. Pentru a verifica dacă un inamic a ajuns în imediata proximitatea a castelului, am implementat în scriptul de comportament al inamicilor (descris in secțiunea imediat următoare) o funcție care este apelată în metoda Update, în care verific distanța rămasă dintre obiectiv (castelul) și poziția unei instanțe de inamic. În momentul în care această distanță scade sub un anumit prag minim, valori precum bugetul, viețile rămase, inamicii ramași în viată (descrise in capitolul I.3.3) vor fi actualizate iar instanța respectivă a inamicului va fi dezactivată.

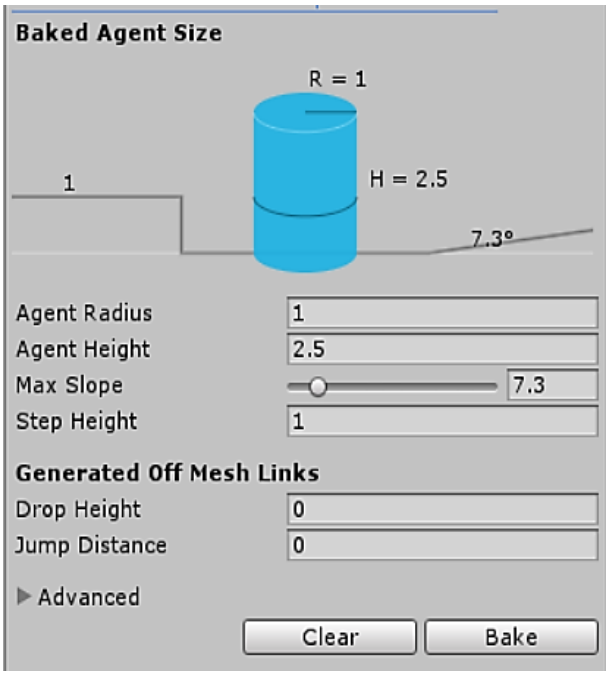


(Figura 12) Baza :castel

Inamicii sunt entități care vor fi generate pe baza unei logici predefinite și vor avea ca scop găsirea bazei jucătorului, fiind capabili să evite obstacolele adăugate în timpul jocului de către jucător (procedeu descris in capitolul I.3.3). La baza implementării inamicilor stă un model 3D căruia îi este atribuit un script care gestionează mișcarea pe hartă, evitarea obstacolelor și evenimentul de eliminare a unei instanțe de inamic. Pentru partea de găsire a drumului și evitare a obstacolelor am folosit unealta pusă la dispoziție de Unity, NavigationMesh.

NavigationMesh pune la dispoziție partea de identificare a zonelor prin care o entitate se poate mișca și un algoritm de găsire a celui mai scurt drum accesibil. Pentru ca acest lucru să fie posibil trebuie definite 2 mari aspecte, “Navigation area” și “Navigation agent”.

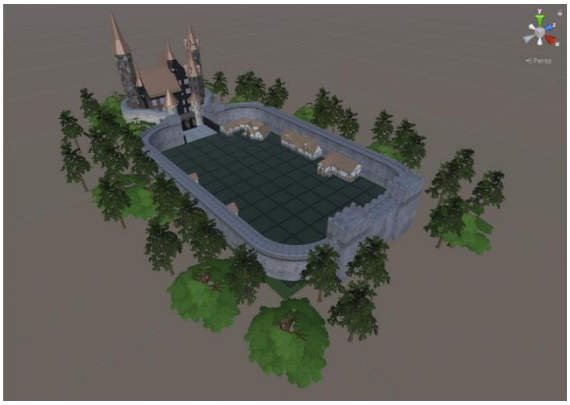
Navigation Agent presupune definirea dimensiunilor și detaliilor de mișcare pentru un anumit tip de inamic. Caracteristicile care pot fi modificate (Figura 12) fiind spațiul circular din jurul inamicului care va fi ocupat de acesta (Agent Radius), înălțimea acestuia (Agent Height), detalii care vor fi luate în considerare la generarea zonei de navigație (mai jos explicate), pentru a determina prin ce zone poate încăpea un astfel de agent. Alte detalii includ mărimea pasului (Step Height) caracteristică necesară determinării zonelor pe care agenții le pot accesa în deplasarea lor pe verticală (ex: scări, mici denivelări de teren), viteza și accelerarea lor (disponibile in componenta scriptului atașat obiectelor de tip inamic). Odată definite detaliile unui astfel de agent, acestuia îi va fi atribuit un tag (identificator unic), pentru a repera in scenă acel tip de inamic, și pentru a deschide posibilitatea de a avea mai multe tipuri de inamici cu caracteristici diferite.



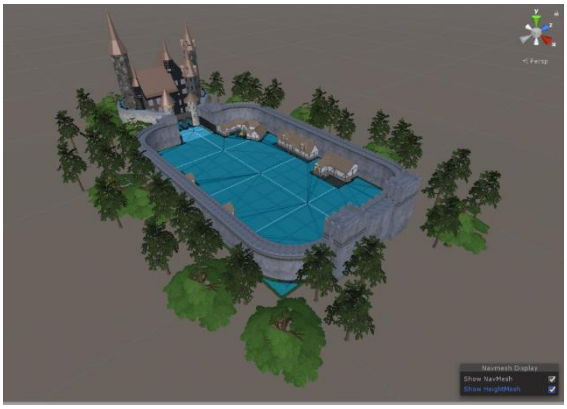
(Figura 12) Caracteristicile unui agent de navigație.

Navigation area presupune marcarea zonelor prin care un agent (o entitate de tip inamic) poate circula, acest lucru fiind ajustabil din Inspectorul aplicației (definit mai sus) unde vom marca obiectele dorite din scenă ca fiind “Navigation static”. Odată marcate, obiectele respective, vom putea genera zona pe care entitățile cu un anumit tag o vor avea la dispoziție pentru navigare. În momentul generării, Unity va detecta toate obstacolele (Figura 14) (zone marcate ca “Navigation static”, prin care inamicii nu pot trece și pe care le vor evita, ex: case, zone de teren în care nu au acces) și de asemenea va detecta structurile defensive plasate pe parcursul jocului de către utilizator. Acest lucru este posibil datorită opțiunilor oferite de Unity, în care se poate specifica ca în momentul în care în scenă va apărea un obiect, (marcat ca Navigation obstacle) acesta va fi considerat obstacol și va actualiza în acel moment zona de navigare. Foarte important este faptul ca această generare trebuie făcută de fiecare data când este realizată o modificare asupra zonei prin care inamicii se vor mișca, pentru ca inamicii să fie conștienți de modificările schimbate și pentru

a se putea calcula drumul în mod corect, bineînțeles cu excepția cazului în care jocul este în desfășurare și jucătorul adaugă un obstacol, după cum am descris mai sus.



(Figura 13) Harta înainte de generarea zonei de navigare



(Figura 14) Harta după ce a fost generata zona de navigare

În cazul aplicației Kingdom HazARd este implementat un singur tip de inamic , controlat de scriptul AgentControll în care gestionez chestiuni precum setarea destinației, acțiunea de a primi daune (scăderea vieții) și câteva îmbunătățiri la calcularea adaptivă a rutei dar și un element de interfață, acela fiind afișarea unei bare care indica viată rămasă a unei instanțe de inamic.

Funcționalitatea de scădere a vieții presupune păstrarea în memorie a unei variabile în care este stocată viața de start care va fi scăzută de fiecare dată când un glonț tras de turnuri va ajunge să atingă inamicul în cauză. În acel moment viața este actualizată și se verifica dacă viața rămasă este un număr mai mare decât zero, caz în care vom actualiza și bara aferentă în mod procentual pentru a face jucătorul conștient de daunele cauzate acelei instanțe de inamic. Având în vedere faptul că bara de viață este un obiect bidimensional pe care a trebuit să îl folosesc într-o scenă tridimensională, a trebuit să ofer ca parametru o referință către punctul de vedere a utilizatorului și să apelez în metoda Update o porțiune de script care orientează bara spre utilizator (Script 3), astfel în orice moment pe parcursul jocului statusul vieții fiecărui inamic va fi vizibil jucătorului.

```
private void Update()
    {
        ArCamera = GameObject.Find("ARCamera");
        healthBarComponent.transform.LookAt(ArCamera.transform);
       ...

    }
```

(Script 3) Ajustarea barei de viță sa fie orientată spre utilizator

I.3.3 Procedee folosite în aplicație

Partea de Tutorial (Script 4) presupune o introducere a jucătorului în mecanismele disponibile în joc. Aceasta parte introductivă este disponibilă la începutul primului nivel și constă dintr-o serie de panouri cu text introductiv în care se prezinta scopul jocului, modalitățile de interacțiune, sistemul de buget și funcționalitățile lui e.t.c. Pentru a restricționa utilizatorul din a interacționa cu mediul primului nivel (încărcat în spatele tutoriaului) am creat un obiect cu opacitate 0 care să intercepteze orice interacțiune a jucătorului cu ceea ce se află în afara panourilor de tutorial. De asemenea pentru a evita orice conflict care ar putea apărea a trebuit să expun variabile precum „`enemyControll.canStartGame`" vizibile din interiorul altor clase întru-cât pot semnala sfârșitul tutorialului și începerea jocului.

```
public void NextPanel()
    {
        switch (currentPanel)
        {
            case 4:
                ListOfPanels[currentPanel - 1].SetActive(false);
                shouldStartTimer = true;
                UIOverlay.SetActive(false);
                break;
            case 11:
                enemyControll.canStartGame = true;
                PausePanel.SetActive(true);
                break;
            default:
                ListOfPanels[currentPanel - 1].SetActive(false);
                ListOfPanels[currentPanel].SetActive(true);
                currentPanel++;
                break;
        }
    `
```

(Script 4) Gestionarea secțiunii de tutorial

Mod de vizualizare/ mod de construcție În mod implicit la intrarea în joc, utilizatorul va fi în modul de vizualizare, în care poate observa întreaga acțiune de pe hartă, fără a putea interacționa cu mediul acesteia în vreun fel. Pentru a putea plasa structuri defensive, utilizatorul poate accesa butonul din dreapta jos al ecranului care îi va activa panoul și matricea de construcții. Ca implementare panoul de contracții are prezent un comutator cu 2 valori, acest lucru însemnând faptul că la un moment dat nu poate fi selectată mai mult de o construcție (zid sau turn). De asemenea, odată cu activarea modului de construcții, se va activa și matricea de control (descrisă mai sus) care îi va permite jucătorului amplasarea de obstacole.

Amplasarea de obstacole

Pentru a realiza amplasarea unui element defensiv pe matricea de construcție, în momentul în care jucătorul va efectua un gest de touch (atingerea ecranului unui dispozitiv ce beneficiază de senzor tactil, echivalent al evenimentului de click de pe calculator) pe ecranul telefonului, evenimentul va fi interpretat în back-end prin generarea uni obiect de tip RaycastHit. Generarea unui astfel de obiect presupune stabilirea unui punct de origine (determinat de punctul în care jucătorul atinge ecranul) și emiterea unei raze perpendiculare pe planul determinat de planul telefonului pe o distanță specificată, în cazul aplicației distanța tinzând la infinit. Obiectul de tip RaycastHit va avea acces după generarea razei (ray) la toate obiectele active din scenă care se găsesc pe traiectoria razei, obiectul de interes principal fiind primul game object atins. Pentru a realiza plasarea unui element defensiv, acesta trebuie să fie de tipul nod, component al matricei de construcție explicată in capitolul anterior (Script 5).

```
void Update(){

        ...

        RaycastHit hit;
         Ray ray;

        if (Physics.Raycast(ray, out hit, Mathf.Infinity) &&
        spawnGrid.activeSelf){

                ...

                if (interactablePlanes[index].transform.name ==
                hit.transform.name){
                 Plane tile = interactablePlanes[index].GetComponent<Plane>();
                 if (!tile.isDisabled){
                        GameObject currentSpawnPoint = spawnPlanes[index];
                    if (objectToBeBuild == wallPrefab{
                          if (currencyLogic.canBuyWall()){
                             currencyLogic.BuyWall();
                             GameObject objectBuildGO =Instantiate(objectToBeBuild,
                             currentSpawnPoint.transform);
                           }
                     }
                    else if (objectToBeBuild == turretPrefab){
                          if (currencyLogic.canBuyTurret()){
                              currencyLogic.BuyTurret();
                              GameObject objectBuildGO=Instantiate(objectToBeBuild,
                               currentSpawnPoint.transform);
                           }
                          }

                  }

                 }

        }

}
```

(Script 5) Amplasarea structurilor defensive

Controlul variabilelor (buget)

In aplicație aproape orice eveniment ce ține de desfășurarea jocului are un cost aferent care este legat direct de buget. Bugetul presupune existența unei sume de bani cu care jucătorul va începe jocul și care se va modifica pe parcurs în funcție de deciziile acestuia. Acțiunile care presupun scăderea bugetului sunt cumpărarea de structuri defensive (ziduri sau turnuri) și îmbunătățirea turnurilor iar modalitatea prin care acesta își poate creste bugetul este prin eliminarea inamicilor folosind turnuri strategic amplasate. De asemenea pe lângă buget, alte variabile prezente în joc sunt cele care țin evidența vieților rămase, a inamicilor care au fost uciși și a inamicilor care sunt în viață și sunt activi pe hartă la un moment dat. Pentru a implementa sistemul de control al variabilelor am folosit design paternul Singleton (Script 6) care îmi asigură existența unei singure instanțe a clasei CurrencyLogic pe care o voi invoca de fiecare dată când unul din evenimentele menționate mai sus se petrece.

```
private static CurrencyLogic currencyLogic;

    private CurrencyLogic(){}

    public static CurrencyLogic getInstance()
    {
        if (!currencyLogic)
        {
            currencyLogic = new CurrencyLogic();
            return currencyLogic;
        }
        else
        {
            return currencyLogic;
        }
    }
```

(Script 6) Singleton pentru sistemul de buget

După cum am menționat, clasa CurrencyLogic gestionează toate aspectele legate de controlul variabilelor, acesta având 4 seturi de metode care ajută la aceasta gestiune. Metodele au fost denumite sugestiv pentru a ajuta la mentenanța aplicației. Categoriile sunt:

Metode care se ocupă de gestiunea directă a bugetului (Script 0.5)

```
public int GetCurrency()
public void AddAmountToCurrency(int amount)
public void DecreaseCurrencyByAmount(int amount)
```

(Script 0.5) Gestiunea bugetului

Metode care gestionează costul construcțiilor și interacțiunea cu bugetul (Script 0.6)

```
public bool canBuyWall()
public int  GetWallCost()
public void BuyWall()
public bool canBuyTurret()
public int GetTurretCost()
public void BuyTurret()
```

(Script 0.6) Gestiunea prețurilor

Metode care se ocupă de gestiunea statutului de în viată/eliminat a inamicilor (Script 0.7)

```
public int GetLifes()
public void DecreaseLifes()
```

(Script 0.7) Gestiunea numărului de inamici

Metode care gestionează viețile rămase ale jucătorului (Script 0.8)

```
public int GetKilled()
public void AddKill()
public int GetAlives()
public void AddAlive()
public void DecreaseAlive()
```

(Script 0.8) Gestiunea vieților jucătorului

Pentru ca jucătorul să își poată apară baza, fluxul de apariție a inamicilor nu va fi unul continuu, din contra, aceștia se vor genera după o logică prestabilită și diferită pentru cele 2 harți disponibile, însă ideea de bază este aceeași la ambele scene. Aceasta presupune inițierea unui cronometru în momentul în care toate condițiile sunt îndeplinite pentru a putea genera un val de inamici, condiții precum partea de tutorial să fie terminată sau dacă un alt val de inamici a mai fost generat, acesta să fi fost eliminat complet (fie de către turnuri, fie să fi ajuns la destinație). În momentul în care acel cronometru va expira, metoda SpawnWave va fi apelată din interiorul metodei Update (Script 7). In interiorul metodei SpawnWave (Script 8) se apelează la un anumit interval (interval coordonat de logica Corutinei (explicații în capitolul 2.3.1)) metoda SpawnAgent (Script 8), responsabilă cu generarea unui singure entități de tip inamic (NavmeshAgent) în care sunt tratate chestiuni precum setarea destinației ca fiind baza jucătorului și actualizarea variabilei corespundatoare (număr inamici în viață) .

```
if (canSpawnNextWave && countdown <= 0f)
        {
            canSpawnNextWave = false;
            if (currentWave < enemiesPerWave.Count)
            {
                StartCoroutine(SpawnWave());
            }
        }
```

(Script 7) Logică de generare a valurilor de inamici

Pentru a mă asigura de faptul ca agentul generat se va află în interiorul zonei de navigație, am implementat o logică în care generez un obiect de tip NavmeshHit (asemănător cu procedeul de plasare a obstacolelor) în care proiectez o rază către zona de navigație și salvez poziția returnată, instanța de inamic urmând să fie generată la acea poziție. La cea de a doua hartă ca și adiție este prevăzută o logică ce alege dintre 2 punte de generare în mod aleatoriu in momentul in care un nou val va fi generat.

```
IEnumerator SpawnWave()
    {
        if(currentWave < enemiesPerWave.Count)
        {
            var numberOfEnemies = enemiesPerWave[currentWave];
            var enemySpeed = enemySpeedPerWave[currentWave];
            canCheckForNextWave = false;

            for (int i = 0; i < numberOfEnemies; i++)
            {
                SpawnAgent();
                yield return new WaitForSeconds(0.5f);
            }
            canCheckForNextWave = true;

            currentWave++;
        }

    }

...

void SpawnAgent()
    {
        currencyLogic.AddAlive();
        currencyLogic.UpdateCurrencyOnDisplay();
        GameObject agentGO = Instantiate(nagaent,
              spawnPoint.transform.position, Quaternion.identity);
        NavMeshAgent navMeshAgent = agentGO.GetComponent<NavMeshAgent>();
        if (navMeshAgent.enabled && !navMeshAgent.isOnNavMesh)
        {
            var position = transform.position;
            NavMeshHit hit;
            NavMesh.SamplePosition(position, out hit, 10.0f, 1);
            position = hit.position;
            navMeshAgent.Warp(position);
        }
    }
```

(Script 8) Implementarea metodei de generare a unui val de inamici si generarea unei instanțe de inamic

Capitolul II. Manual de utilizare

II.1 Cuvânt înainte

Această secțiune este destinată persoanelor care doresc să interacționeze cu aplicația din punct de vedere al unui jucător. Aplicația va solicita acces la funcționalitatea de cameră a telefonului pentru a putea fi rulată, așa ca utilizatorul va trebui să permită acest aspect prima dată când va intra in aplicație.

Având în vedere faptul că aplicația este un joc destinat terminalelor mobile cu sistem de operare Android, orice interacțiune se va face pe baza evenimentului de touch (atingere) a ecranului telefonului.

Valorile care apar în poze (buget/ vieți rămase/ prețul structurilor defensive/ prețul îmbunătățirii turnurilor) sunt cu caracter informativ ce țin de mediul de dezvoltare. Versiunea finală a aplicației beneficiază de valori balansate care duc la o experiență plăcută a jocului.

II.2 Prima interacțiune cu aplicați

În momentul în care utilizatorul va porni aplicația, va fi întâmpinat de un meniu simplist care ii pune la dispoziție 2 opțiuni foarte explicite: (Manual 1) selectarea unui nivel sau ieșirea din aplicație.

(Manual 1) Meniul principal al aplicației

În momentul în care jucătorul dorește să aleagă un nivel și apasă pe butonul „LEVEL SELECT” acesta va avea 2 opțiuni. O hartă de dimensiuni relativ reduse care beneficiază și de partea de tutorial a jocului, aceasta ilustrând o micuța așezare rurala, a doua opțiune fiind un platou montan de dimensiuni considerabil mai mari și cu un nivel de dificultate mai ridicat.

(Manual 2) Selectarea nivelului

II.3 Etapele jocului

În cele ce urmează voi face o prezentare a mecanismelor prezente în joc și a posibilităților pe care un jucător le are în aceasta aplicație, exemplificând aceste mecanisme folosindu-mă de primul nivel al jocului, pentru a nu expune întreaga aplicație în acest manual de utilizare.

II.3.1 Începerea jocului

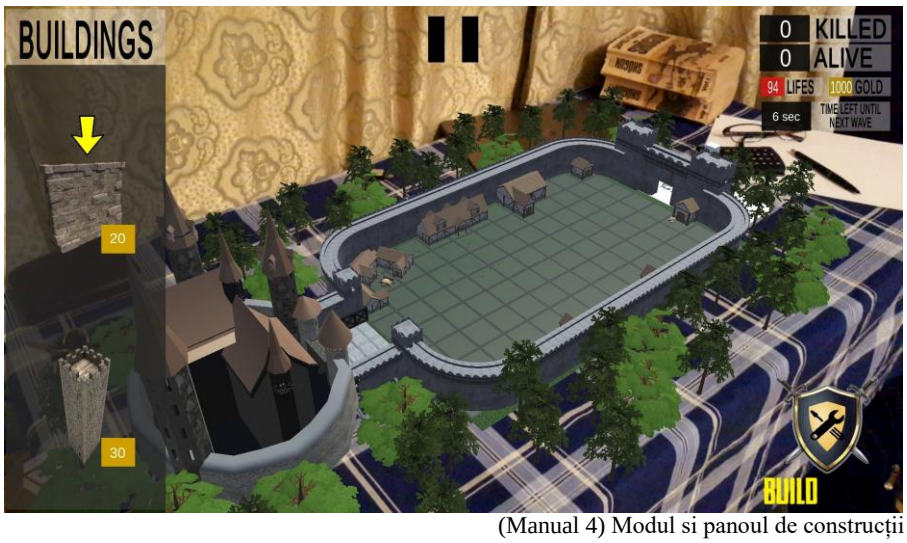
După ce utilizatorul a selectat un nivel, acesta va fi redirecționat către nivelul corespunzător. Jucătorul trebuie să îndrepte camera telefonului spre imaginea țină, așezata pe o suprafață dreaptă, de preferat în condiții de iluminare cât mai optime. În momentul în care aplicația va detecta imaginea țină, harta va fi afișată (Manual 3) și jocul poate începe. În cazul primului

nivel, utilizatorul va fi nevoit să treacă prin partea de tutorial în care îi sunt explicate aspecte esențiale ale aplicației.

(Manual 3) Meniul principal al aplicației

II.3.2 Desfășurarea Jocului

Din acest moment, jocul este în desfășurare și scopul jucătorului este ca în orice joc de tip Tower Defense să își apere baza. (castelul, în cazul de față) Pentru a putea adaugă structuri defensive în scenă, jucătorul va trebui să apese pe scutul din partea dreapta-jos a ecranului pentru a intra în modul de construcții. (Manual 4)



(Manual 4) Modul si panoul de construcții

În momentul în care utilizatorul intră în modul de construcție, 2 elemente noi apar în scenă, acestea fiind matricea de construcție și meniul de structuri defensive disponibile. În meniul apărut, clădirea selectată pentru a fi construită este marcată cu ajutorul unei săgeți (in mod implicit la prima deschidere fiind selectat zidul), jucătorul fiind capabil să își schimbe oricând alegerea. La baza imaginilor din meniul de construcții este afișat prețul acelor structuri, suma care fi scăzută din bugetul jucătorului dacă acesta alege să construiască acea structura. Dacă utilizatorul dorește să plaseze o structură defensivă în scenă, acesta va selecta una din cele doua imagini din meniu și va atinge pe ecran zona de pe matricea de construcție în care dorește să fie amplasată.

Plasarea de ziduri este un element de originalitate în acest joc, utilizatorul fiind capabil să-și creeze propriul labirint prin care inamicii vor circula (Manual 5 și Manual 6). Diferite strategii de abordare a plasării de ziduri:

(Manual 5) Plasare a zidurilor in mod haotic

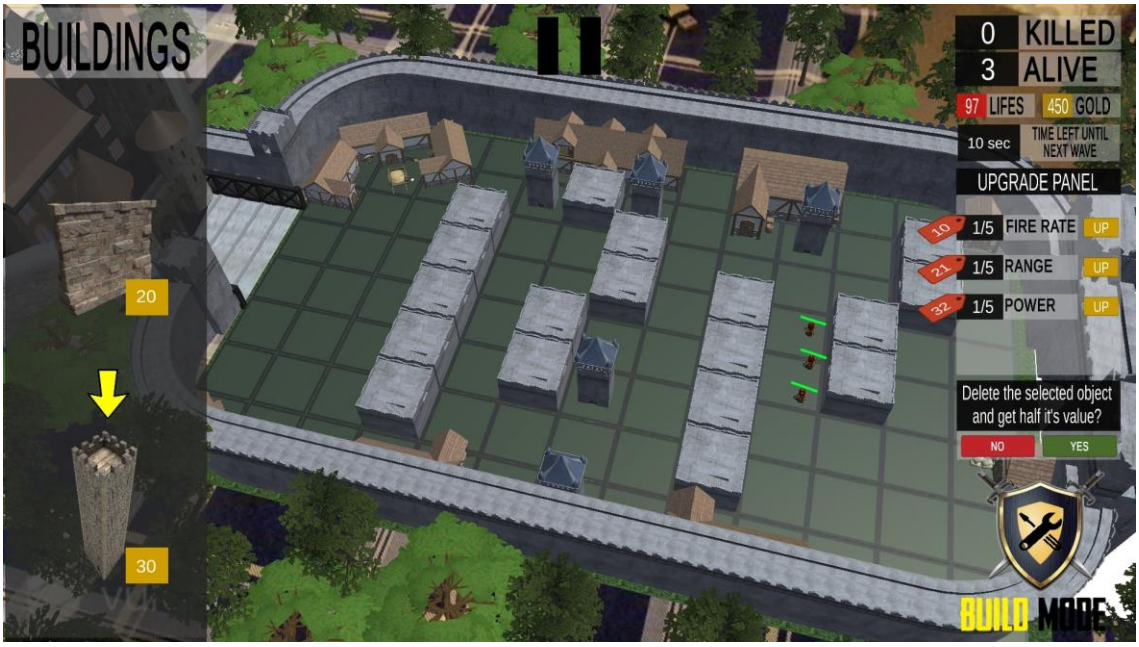
(Manual 6) Plasarea zidurilor după o strategie

Amplasarea zidurilor se face fără nicio restricție atâta timp cât aceasta acțiune nu va duce la cazul în care inamicii nu pot avansa în drumul lor spre bază. În cazul în care jucătorul va încerca totuși să blocheze calea inamicilor, aplicația ii va distruge structura care a dus la blocaj fără ca jucătorul să își primească suma cheltuita pe ea. (Manual 7)

(Manual 7) Imposibilitatea de a bloca drumul inamicilor

Plasarea de turnuri presupune amplasarea acestora în zone astfel încât să acopere o rază de tragere cât mai mare. După cum am menționat și în capitolul **2.3.2**, turnurile nu vor bloca întreaga zona în care sunt amplasate, deci nu este indicat folosirea lor pentru ghidarea valului de inamici. Tot în capitolul **2.3.2** am specificat faptul că turnurile pot fi îmbunătățite contra unui cost (Manual 8).

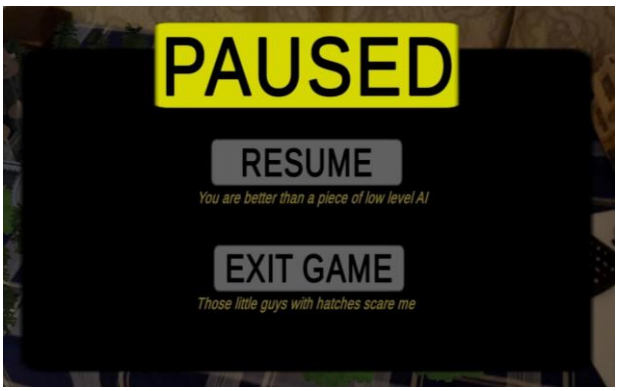
Aspectele care pot fi îmbunătățite la un turn țin de viteza de tragere („fire rate”), raza în care un turn poate detecta inamici(„range”) și puterea gloanțelor în momentul în care acestea lovesc un inamic („power”). În funcție de amplasarea turnurilor sale, un jucător se poate axa pe unele dintre aceste aspecte mai mult decât pe altele, alegerea sa fiind liberă. Aceste îmbunătățiri fiind disponibile individual, desigur vor avea și prețuri individuale. Pentru a cumpăra îmbunătățiri pentru un turn, jucătorul trebuie să fie întrat în modul de construcție și să selecteze în joc turnul dorit, în acel moment, un meniu adițional va apărea în partea dreapta a ecranului. Pentru a finaliza achiziția, jucătorul va apasă pe butonul „UP” din dreptul îmbunătățirii dorite, preturile fiind afișate în partea stângă a fiecărei îmbunătățiri.



(Manual 8) Meniul de îmbunătățiri a turnurilor si meniul de eliminarea a uni structuri

Valabil pentru ambele structuri defensive, în momentul în care jucătorul va selecta o construcție plasată anterior, acesta va avea opțiunea să o elimine, primind înapoi jumătate din prețul achitat pe ea. Această acțiune este posibilă prin intermediul meniului adițional care va fi disponibil în partea dreaptă a ecranului. (Manual 8)

Jucătorul are de asemenea posibilitatea de pune pauza execuției jocului, astfel toată acțiunea incluzând mișcarea inamicilor și mecanismul de tragere a turnurilor vor fi blocate și un meniu de pauza (Manual 9) va apărea pe ecran. În momentul în care acesta va apela meniul de pauză folosind butonul din partea superioară a ecranului, acestuia îi va apărea meniul din imaginea “Manual 9” în care opțiunile pe care le are sunt de a relua jocul din punctul de unde a rămas sau de a se întoarce la meniul principal al aplicației. (Manual 1)



(Manual 9) Meniul de pauză

II.3.3 Încheierea jocului

Un nivel poate fi încheiat în 2 modalități. Fie prin eliminarea tuturor valurilor de inamici dintr-un nivel, caz în care jucătorul a câștigat acel nivel (Manual 10.2) sau daca rămâne fără vieți (au pătruns în castel numărul limită de inamici) , caz în care jucătorul a pierdut acel nivel (Manual 10.1).

(Manual 10.1) Ecranul de Înfrângere

(Manual 10.2) Ecranul de victorie

Capitolul III. Îmbunătățiri

Aplicația Kingdom HazARd este încă un pas în lumea jocurilor AR, lume care este încă la început dar în continuă expansiune.

În stadiul final, această aplicație acoperă toate elementele de bază ale unui joc Tower Defense, însă acest aspect lasă o mulțime de oportunități pentru viitoare îmbunătățiri. Noi harți pot fi adăugate, noi tipuri de inamici care să aibă diferite caracteristici (viață/ viteză/ durabilitate), o nouă abordare a jocului în care în loc de construcții, jucătorul va folosi puteri care să aibă diferite efecte asupra inamicilor.

În partea de optimizare, inteligența artificiala (Navmesh Agent) din Unity poate fi înlocuită cu un algoritm de găsire a celui mai scurt drum care să ofere o complexitate mai mică și un timp mai bun de răspuns, ceea ce ar duce la o mișcare a agenților mult mai fluentă.

Poate fi adăugată o legătura cu o bază de date care să faciliteze înregistrarea unui jucător în aplicație și salvarea progresului sau chiar și după ce acesta a ieșit din aplicație, acest aspect fiind volatil în momentul de față, progresul fiind salvat doar cât jocul este în execuție.

Concluzii

Pe parcursul dezvoltării acestui proiect am reușit să îmi satisfac nemulțumirea care m-a determinat să aleg această temă de licența, aceea fiind lipsa de conținut pe piața jocurilor AR. Am reușit de asemenea să înțeleg la un nivel mai avansat posibilitățile pe care le oferă Unity și să mă familiarizez cu un SDK disponibil pentru acest program. Am reușit să aduc soluții creative în momente în care anumite funcționalități pe care doream să le implementez nu aveau un suport dedicat și a trebuit să improvizez, dar cel mai important, am reușit să lucrez la un proiect cu plăcere învățând lucruri noi, care la început păreau mult peste nivelul meu de cunoaștere, dovedindu-mi că singurele limite pe care le-am avut de depășit, au fost cele pe care, inconștient, mi le-am impus de unul singur.

Bibliografie

1. Informații statistice privind piața jocurilor video
   https://www.theatlantic.com/technology/archive/2011/06/infographic-video-game-industry-statistics/239665
2. Documentația oficiala a limbajului C#:
   https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide
3. Documentația oficiala a limbajului Boo si detalii despre acesta:
   https://boo-language.github.io
4. Documentația oficiala Unity:
   https://unity3d.com/learn/tutorials/s/scripting
5. Documentația oficială a librăriei Mono:
   thttp://www.mono-project.com
6. Documentație privind librăria Mono:
   http://docs.unity3d.com/ScriptReference/MonoBehaviour.html
7. Portal al dezvoltatorilor oferit de Vuforia:
   https://developer.vuforia.com
8. Informații despre Vuforia SDK:
   https://gravityjack.com/news/vuforia-sdk-gravity-jacks-browsar-code-stack-is-an-augmented-reality-developers-dream
9. Sursa de fișiere audio fără drepturi de autor:
   https://freesound.org
10. Sursa de modele 3D gratuite, oferite de Unity:
    https://assetstore.unity.com/lists/free-assets-34756
11. Sursa de modele 3D gratuite, fără drepturi de autor:
    https://itch.io/game-assets/free
12. Tutoriale urmărite:
    https://app.pluralsight.com/library/courses/introduction-game-development-unity
    https://app.pluralsight.com/library/courses/unity-vuforia-building-ar-experience
13. Informații despre genul de joc TD:
    https://www.giantbomb.com/tower-defense/3015-413
14. Informații despre realitatea augmentată:
    http://www.realitytechnologies.com/augmented-reality

Anexa 1.Tehnologii folosite

A 1.1 Unity

A 1.1.1 Descrierea programului

Unity este un motor de dezvoltare a jocurilor folosit atât pentru dezvoltare de aplicații bidimensionale (2D) cât și tridimensionale (3D) care pune la dispoziție dezvoltatorului posibilitatea de a crea simultan jocuri pentru mai multe platforme, cum ar fi desktop, mobile, console și SmartTV. Prima dată fiind prezentat doar ca un program destinat strict dezvoltării IOS la Conferința destinată dezvoltatorilor, ținută de Apple în 2005 a ajuns 13 ani mai târziu să suporte dezvoltarea aplicațiilor pentru 27 de platforme, în această perioadă nu mai puțin de 6 versiuni de Unity fiind lansate.

A 1.1.2 Mediul de dezvoltare

Unity este un program multifuncțional care suportă grafici de atât de tip 2D cat și 3D plasate într-o scenă , folosind ca modalitate de interacțiune principala metoda “drag and drop” și pe parte de programare limbajul C#, în trecut Unity suportând alte 2 limbaje de programare, acestea fiind UnityScript și Boo. UnityScript a început să fie din în ce mai puțin folosit în ultimii ani, ne mai fiind suportat din August 2017 odată cu lansarea versiunii Unity 2017.1. pe când suportul din partea celor de la Unity pentru limbajul Boo s-a oprit complet încă din 2014.

Limbajul C#

În prezent este alegerea principală când vine vorba de programare în Unity, datorită suportului imens de care are parte acest limbaj. Întreaga documentație oficială este scrisă în C#, majoritatea dezvoltatorilor îl folosesc, deci există o mare posibilitate ca problemele cu care te întâlnești ca dezvoltator să fi fost deja discutate și rezolvate de comunitate. Pe lângă aceste aspecte, C# beneficiază de multitudinea de librarii disponibile care fac integrarea cu alte module de aplicații mult mai ușoară.

UnityScript

Unity Script este implementarea celor de la Unity a unui limbaj de programare cu sintaxa bazată pe Java Script. La fel ca și Java Script, Unity Script este un limbaj netipizat care oferă posibilitatea de încapsulare și beneficiază de un număr decent de frameworkuri ajutătoare dar care nu se ridica la nivelul complexității C# - ului.

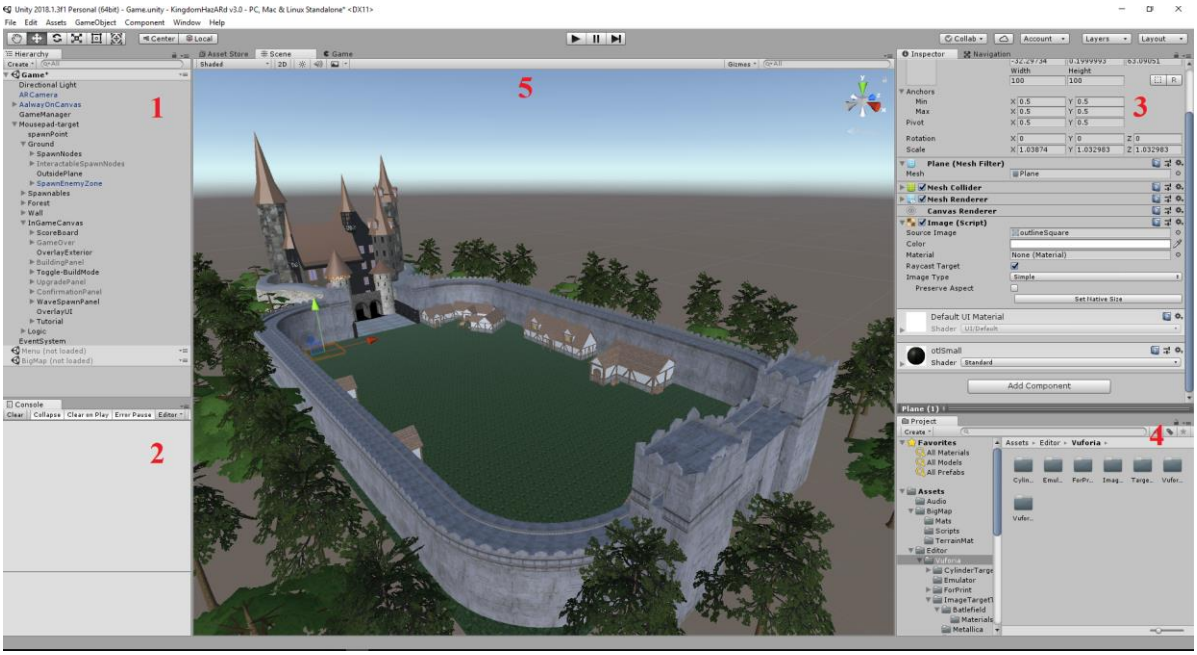
Boo

Este un limbaj de programare static apărut în anul 2003 ce are la baza sintaxa limbajului Python. Acesta a fost introdus ca limbaj de programare în Unity pentru a mari numărul de dezvoltatori interesați de aceasta piața. Motivul pentru care Boo nu a cunoscut niciodată popularitatea C#-ului sau Unity Script-ului este lipsa de documentație vizavi de acest limbaj și lipsa de unelte ajutătoare care să faciliteze o dezvoltare de aplicații mai complexe. Unity au oprit complet suportul pentru Boo în 2014 dar l-au păstrat integrat ca limbaj de programare pentru cei care aveau aplicații deja în producție și aveau la baza acest limbaj.

Ca element de bază în interacțiunea dintre dezvoltator și Unity se afla entitățile care poarta denumirea de “game object”. Unui astfel de obiect i se pot modifica proprietăți precum amplasarea, rotația, culoarea, i se pot atribui scripturi care să dicteze un anumit comportament . “game object” - urile pot avea o multitudine de înfățișări precum: cuburi, sfere, plane, cilindre sau chiar modele 3D predefinite. Pentru a fi vizibil în scena și implicit în joc, game objectul trebuie să fie activ, în caz contrar proprietățile atribuite acelui obiect nu vor avea nicio influentă asupra scenei.

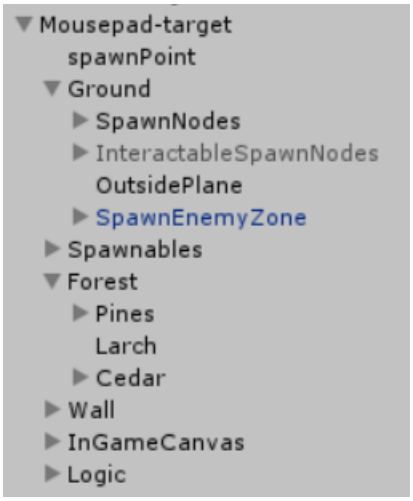
A 1.1.3 Interfața

Interfața programului se împarte în 5 zone principale(Figura A1), destinate interacțiunii cu obiectele folosite în proiect, cu proprietățile acestora dar și cu zonele de pe hard disk unde sunt stocate imagini, scripturi, fișiere audio etc.



(Figura A1) Interfața programului Unity

1.Ierarhia proiectului(Hierarchy)



(Figura A2) Ierarhia în Unity

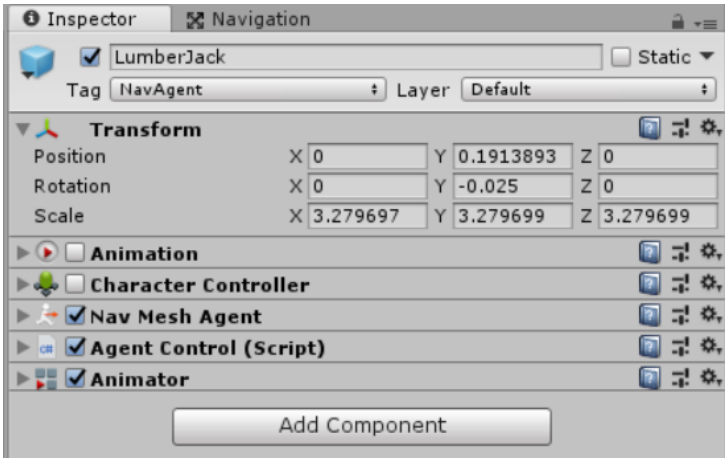
Ierarhia proiectului în Unity se bazează pe relații de tip partite-copil unde un copil poate avea un singur părinte și un părinte poate avea mai mulți copii (Figura A2). Odată asignat unui obiect părinte, obiectul copil va avea o poziționare relativă față de acesta, astfel atunci când dezvoltatorul va altera proprietăți ale obiectului părinte, (locație, mărime, rotație) acestea se vor propaga și asupra tuturor copiilor. În cazul complementar în care alteram o proprietate asupra obiectului copil, aceasta modificare nu va influența obiectul părinte. Este de preferat ca atunci când dezvoltam o aplicație în Unity să grupam corespunzător obiectele în ierarhie astfel încât să avem un control cât mai bun asupra obiectelor și o precizie cat mai mare asupra detaliilor și proporțiilor.

2.Consola dezvoltatorului(Console)

Consola servește rolul de a ajuta dezvoltatorul prin a-i aduce la cunoștință anumite erori sau avertismente (eng. warnings) care pot apărea fie din cod fie din scena (datorită anumitor obiecte din scena)

3.Inspectorul de elemente (Inspector)

În inspector putem accesa, atribui și altera proprietăți ale obiectelor. În această zona a interfeței putem modifica elemente precum amplasarea în scena a obiectelor, dimensiunea lor, logica de mișcare, putem adaugă coliziuni (celelalte obiecte se vor bloca atunci când vor atinge în calea lor obiectele cu coliziuni) sau chiar adaugă elemente de inteligenta artificiala care să ghideze mișcarea obiectelor.



(Figura A3) Inspectorul din Unity

4.Zona de resurse (Project)

În zona de resurse a proiectul regăsim toate fișierele (imagini, fișiere audio, modele 3D) care vor fi importate și vor interacționa cu aplicația dezvoltată)

5.Panoul de Scena/Joc (Scene/Game Panel)

Panoul scenă este locul în care toate obiectele sunt puse împreună pentru a forma acțiunea jocului. În momentul în care obiectele au fost puse în scenă și au fost modificate în inspector după nevoile dezvoltatorului, acesta poate porni scena respectivă, astfel activând panoul de Joc în care va fi simulat comportamentul jocului conform logicii construite până în acel moment.

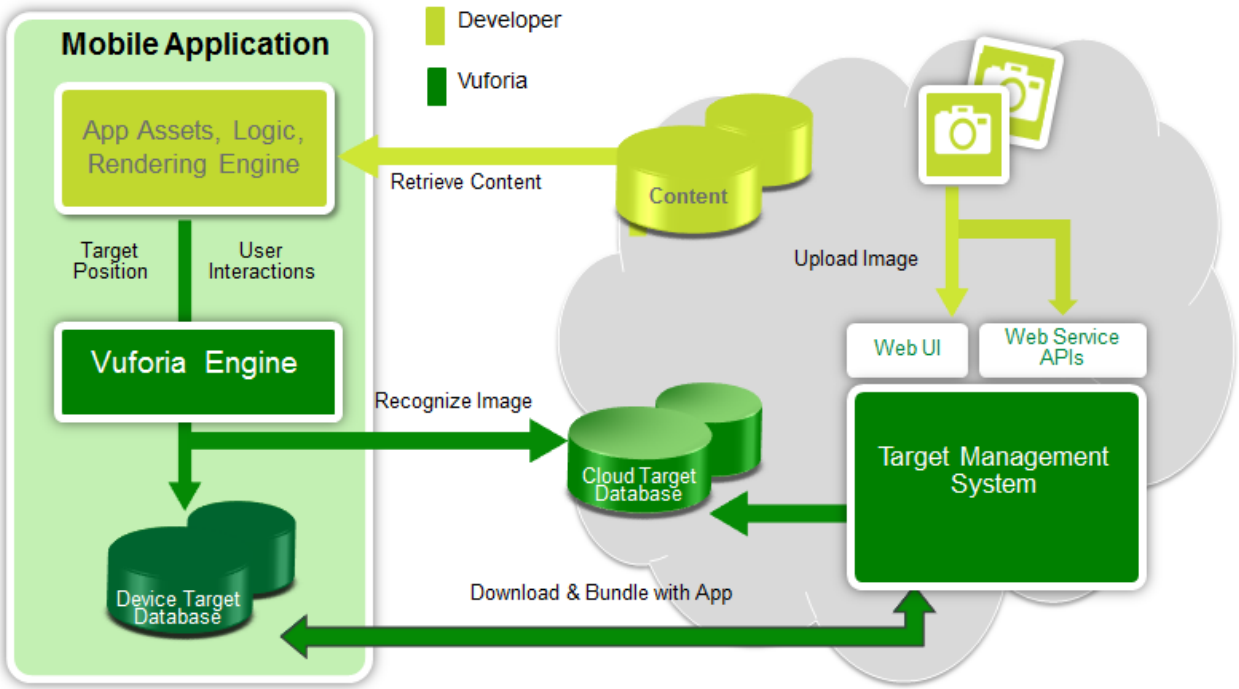
A 1.1.4 Librăria Mono

Programarea în Unity este bazată pe librăria Mono, susținută de cei de la Xamarin, aceasta fiind o implementare de tip open-source (codul este public și oricine poate aduce îmbunătățiri) a frameworkului .NET bazata pe standardele ECMA pentru C# și CLR(Common language runtime). Datorită faptului ca această librărie este open-source comunitatea de dezvoltatori care contribuie la îmbunătățire și mentenanța este numeroasă și acest lucru asigură siguranță și suport continuu.

A 1.2 Vuforia SDK

A 1.2.1Descrierea SDK-ului

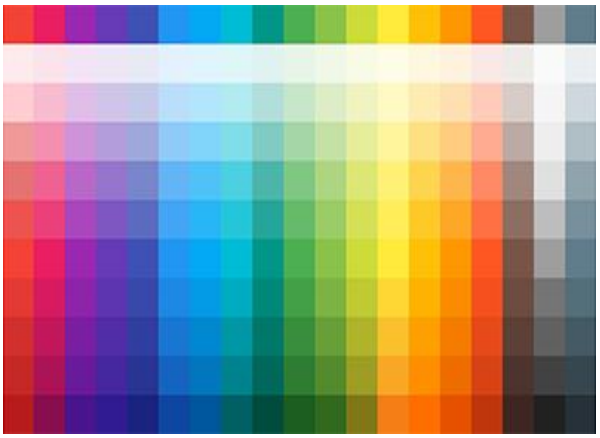
VuForia SDK este folosit pentru facilitarea lucrului cu mediul Realității augmentate. Pentru a realiza detectare cât mai precisă a lumii reale, SDK-ul se folosește de diverse mecanisme de legătura, cum ar fi: „Image-target” , „Object-Target”, „Plane Find Detection”.



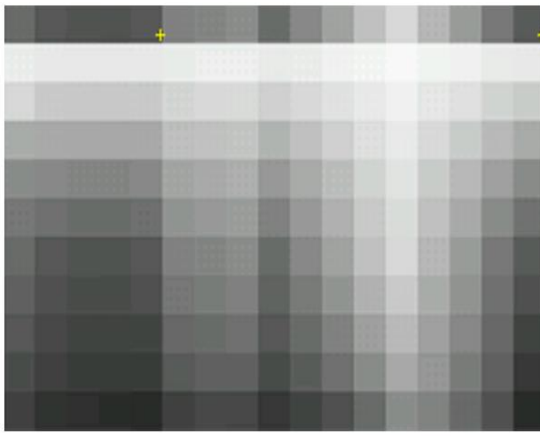
(Figura A4) Modul in care Vuforia accesează tinetele înregistrate

A 1.2.2Legatura cu lumea reală

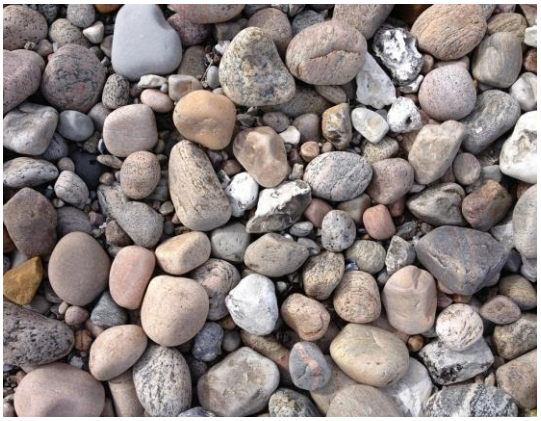
Image-target are la origine o imagine la alegere care suferă un proces de transformare pentru a putea fi importata în Unity și integrată în proiect. Pentru a crea un „image-target” este necesar un fișier de tip .png sau .jpeg cu o dimensiune maximă de 2Mb care va fi încărcat pe site-ul pus la dispoziție de cei de la Vuforia în care imaginea va fi mai întâi convertita din formatul ei original în format alb-negru după care imaginea va fi analizată. Analiza imaginii constă în identificarea punctelor de interes din imagine (un punct de interes înseamnă marcarea unei zone distincte din imagine care ar putea fi identificată de către aplicație,) și în functie de complexitatea imaginii (numarul de puncte de interes) imaginea va primi un rating între 0 și 5 stele, unde o stea reprezinta o detectie slaba pe cand 5 stele asigura o faptul ca imaginea va oferi o detectie foarte buna(cu cât o imagine are mai multe stele cu atat este mai augemntabilă). Dupa realizarea transformarii dintr-o imagine la alegere în „image-target”, utilizatorul poate grupa mai multe astfel de obiecte în foldere, foldere care pot fi descarcate ulterior ca baze de date, importate în Unity și folosite în proiect. O baza de date poate contine un minim de un image-target și poate fi folosita în mai multe proiecte în acelasi timp.



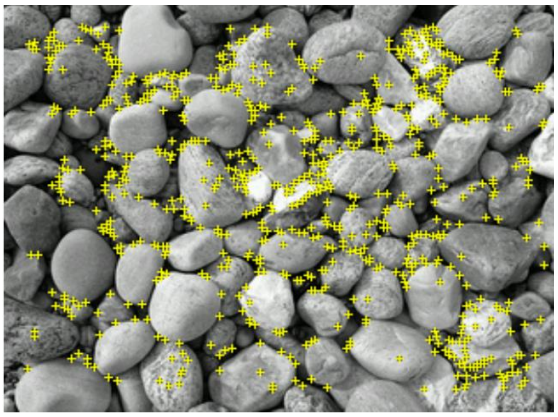
(Figura A5.1) Imagine originală



(Figura A5.2) Imagine procesată, argumentabilitate 0/5

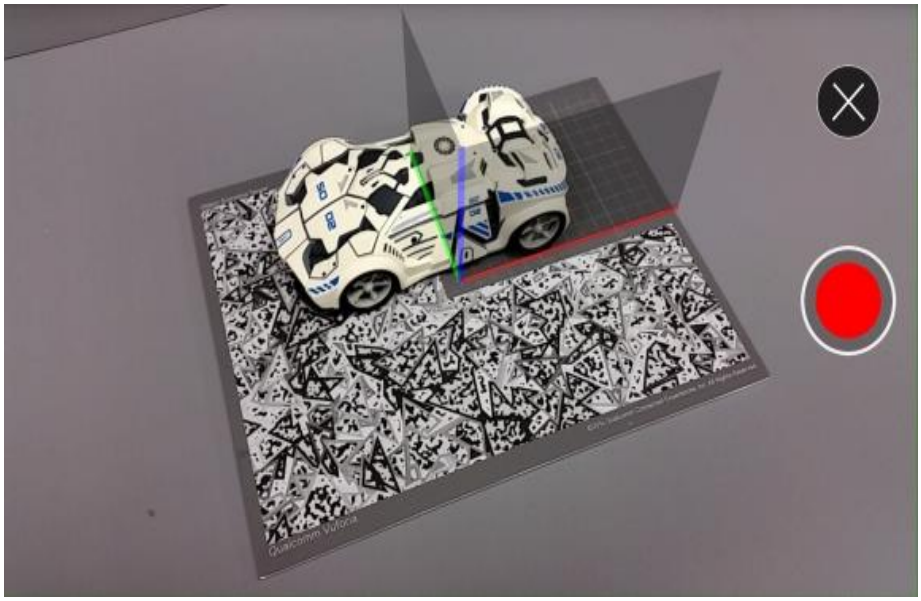


(Figura A5.3) Imagine originală



(Figura A5.4) Imagine procesată, argumentabilitate 5/5

Object-target este asemănător ca și comportament cu „image-target” diferențele fiind modul de generare și faptul ca „Object-target” este un obiect 3D din lumea reală. Generarea se face tot prin intermediul site-ului Vuforia, utilizatorul trebuind de această dată să folosească camera telefonului pentru a realiza o analiză 360 a obiectului (Figura A6), obiect care trebuie plasat pe un model (care trebuie imprimat) găsit desemna pe site. Managementul și folosirea ulterioară a obiectelor este identica cu cea menționata mai sus în cadrul imaginilor țintă.



(Figura A6) Analiza unui obiect pentru a fi transformat in object-target

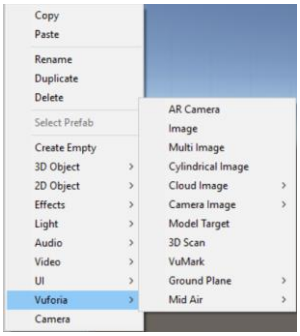
Plane Find Detection este un procedeu relativ nou introdus în Unity 2017.2 prin care SDK-ul este capabil să detecteze o suprafață plană din lumea reală, cum ar fi o masă, podeaua sau un drum stradal fără a mai avea nevoie de o referința predefinita de utilizator pentru a genera conținut virtual ca și în cazurile precedente.

(Figura A7.1) Cadru din lumea reală al unei străzi

(Figura A7.2) Generarea unui model 3D pe planul detectat

A 1.2.3 Integrarea cu Unity

În trecut Vuforia era disponibila dezvoltatorilor sub forma de pachet extern, dar din versiunea Unity 2017.2 Vuforia vine integrat în instalarea programului. Pentru a putea folosi capacitățile SDK-ului, dezvoltatorul va interacționa cu obiecte din categoria Vuforia, corespunzătoare elementelor de realitate augmentată(Figura A8).



(Figura A8) Introducerea în scenă a unui obiect aparținând de Vuforia