

Serenity Garden TD

Opria Ion-Bogdan

January 14, 2021

Contents

1 Introducere	3
1.1 Descrierea proiectului	3
1.2 Introducere in Game Development	4
1.2.1 Scurta istorie	4
1.2.2 Industria curenta	4
2 Introducere in Unity	4
2.1 Prezentare generala	4
2.1.1 GameObject si Mesh	4
2.1.2 Collider	4
2.1.3 Transform si Rigidbody	4
2.1.4 API-ul de scriptare	4
2.2 Motivul alegерii pentru proiectul de licenta	4
2.3 Elemente specifice folosite	4
2.3.1 Shader Graph	4
3 Planificare Proiect	4
3.1 Faza de Proiectare	4
3.2 Tematica Jocului	5
3.3 Caracterele Jocului	5
3.3.1 Inamicii	5
3.3.2 Turnuri defensive	6
3.4 Sistemele Jocului	9
3.4.1 Mapa	9
3.4.2 Sistemul Turnurilor	11
3.4.3 Comandantul	11
3.4.4 Tura de lupta (enemy wave)	11
3.4.5 Nivelele jocului	12

3.4.6	Lock-on	12
3.4.7	Magazinul de upgrade-uri permanente	12
3.4.8	Raid system	13
3.5	Analiza proiectului din punctul de vedere al consumatorilor	13
3.5.1	Dificultatea jocului	13
3.5.2	Elemente de dependenta	14
3.5.3	Grupele de varsta vizate	14
4	Implementare proiect	14
4.1	Scena de lupta	14
4.1.1	Initializarea scripturilor	14
4.1.2	Procesarea comenziilor venite de la utilizator	16
4.1.3	Grid system	16
4.1.4	Sistemul de navigare	16
4.1.5	Ierarhia inamicilor	18
4.1.6	Ierarhia turnurilor defensive	18
4.2	Selectarea nivelelor	18
4.3	Magazinul de upgrade-uri permanente	18
4.4	Sistemul co-op	18
5	Concluzii	18

1 Introducere

1.1 Descrierea proiectului

Jocurile video au inceput sa acapareze din ce in ce mai mult vietile noastre datorita usurintei de accesibilitate si a experientei pe care o ofera. Odata cu evolutia hardware-ului, jocurile video au devenit din ce in ce mai realiste si prin urmare ofera experiente care nu poate fi gasite in niciun alt mediu existent, deoarece in nici un alt mediu nu putem controla si traii experientele unor caractere atat de indetaliat. Industria jocurilor video este o industrie care este in continua crestere si prin urmare este o ramura care merita explorata din perspectiva unui programator si nu nu mai.

”Games are great to work on because they are as much about art as they are science.” [1]

Din aceste motive, am ales drept proiect pentru licenta sa fac un joc video. Jocul se numeste ”Serenity Garden” si este un tower defense 3D. Experienta jocului are loc pe niste nivele special definite, in care utilizatorul trebuie sa isi projezeaza baza. Există multe tipuri diferite de inamici care vor sa distruga baza player-ului, iar acesta, pentru a o proteja, trebuie sa construiasca un sistem defensiv pentru a ii tine la distanta. Player-ul poate sa construiasca turnuri defensive care au diferite efecte asupra inamicilor. Inamicii ataca tot ce gasesc in cale, iar daca reusesc sa distruga baza, player-ul pierde nivelul curent.

Player-ul nu poate sa construiasca turnuri oriunde, ci are locuri predefinite unde poate sa le construiasca, locuri specificate de blocuri hexagonale pe mapa. Odata plasata o tureta, aceasta nu poate fi mutata, dar player-ul are si un caracter invulnerabil pe care il poate muta oriunde doreste pe mapa. Acest caracter va ataca inamicii automat cand nu se afla in miscare, si poate intra in turete pentru a le creste puterea/eficienta.

Modul care va differentia acest joc de celealte jocuri pe acest stil este modul de co-op. 2 jucatori se vor putea conecta prin retea si vor juca un nivel dificil care necesita cooperare si o planificare buna intre ei.

1.2 Introducere in Game Development

1.2.1 Scurta istorie

1.2.2 Industria curenta

2 Introducere in Unity

2.1 Prezentare generala

2.1.1 GameObject si Mesh

2.1.2 Collider

2.1.3 Transform si Rigidbody

2.1.4 API-ul de scriptare

2.2 Motivul alegerii pentru proiectul de licenta

2.3 Elemente specifice folosite

2.3.1 Shader Graph

3 Planificare Proiect

3.1 Faza de Proiectare

”Almost anything that you can be good at can become a useful skill for a game designer.” [2]

Din proiecte precedente am realizat ca o planificare buna a unui proiect inca de la inceput poate imbunatatii calitatea proiectului exponential, asigura usurinta de adaugare a unor elemente noi (fara sa fie necesara rescrierea/re-organizarea proiectului) si reduce frustrarea programatorilor care lucreaza la proiect.

Din acest motiv, inca de la inceputul proiectului am incercat sa imi planific cat mai indetaliat continutul acestuia si modul in care il voi implementa.

In prima faza, mi-am scris un design document care continea descrierea si elementele jocului, privite din multe perspective.

Introducerea proiectului a fost realizata la inceputul acestui document si nu va fi reluată aici.

3.2 Tematica Jocului

Jocul are loc in viitorul departat. Planeta ta este supra-populata si nivelul de poluare a ajuns la un nivel critic. Umanitatea a trecut de mult de punctul in care mai pot salva aceasta planeta. Intreaga planeta va muri in cateva decenii, iar oamenii daca nu isi gasesc o alta casa intre timp, vor muri si ei cu ea. In acest scop, o armata speciala a fost formata, care are ca scop explorarea altor planete si gasirea unei planete care poate suporta rasa humana.

Universul este totusi foarte periculos. Unele planete au viata extraterestra foarte agresiva, iar altele au fost distruse de experimente cibernetice esuate. Player-ul este comandantul suprem al acestei armate care are ca scop protejarea oamenilor de stiinta care vor analiza aceste planete.

Prima faza a proiectului va contine o singura planeta care poate fi populata dar care contine multi inamici periculosi.

Planeta mama trimite resurse din cand in cand, dar numai un numar limitat de resurse pot fi trimise deodata, iar acestea dureaza mult timp sa ajunga, asa ca nu putem depinde de ele in timpul unei lupte.

3.3 Caracterele Jocului

Comandantul este protagonistul jocului. El este caracterul din a carui perspectiva vom juca jocul. A fost cobaiul unui experiment menit sa creeze super-soldati, iar in urma acestui experiment a primit o putere speciala. Poate sa isi ascunda complet prezenta fata de alti oameni/alte creaturi. Din acest motiv, el este caracterul invulnerabil pe care il putem misca pe mapa in lupta.

Locotenentul este ajutorul tau numarul 1. El/ea iti va prezenta tutorialele, iti va da sfaturi si va juca un rol vital in poveste, in caz ca proiectul ajunge in acest punct.

3.3.1 Inamicii

- **Melee** este un tip de inamic care poate ataca turetele doar cand sta fix in fata lor. Are viata mai multa decat inamicii "ranged".
- **Ranged** este un tip de inamic care poate ataca turetele de la distanta. Are viata putina, dar pot ataca repede.
- **Flying** este un inamic care poate fi lovit doar de un numar limitat de turete.
- **Ambusher** este un tip de inamic care ignora turnurile din cale. Acesta se misca pe cel mai scurt drum catre baza player-ului, iar daca ajunge

deasupra acesteia, va lansa bombe care ii vor scadea drastic viata, dupa care vor pleca de pe mapa. Este un tip de inamic zburator care poate fi lovit doar de un numar limitat de turnuri.

Acesti inamici (cu exceptia ambusher-ului) se vor misca pe mapa calculand cel mai scurt drum catre baza player-ului. Daca acestia intalnesc in cale un turn, se vor opri si vor ataca acel turn. Cand turnul este distrus, ei isi vor relua drumul.

Pentru fiecare inamic vor fi 3 tipuri de variatii, fiecare mai puternic decat cel precedent.

3.3.2 Turnuri defensive

In continuare voi scrie o descriere scurta pentru fiecare turn din joc. Toate proprietatile despre care urmeaza sa vorbesc sunt in comparatie cu celelalte turnuri defensive.

- **Serenity** este baza playerilor. Are viata foarte multa daca este distrusa player-ul pierde jocul si poate ataca inamicii care se apropie de ea. Are raza de atac medie, putere de atac medie, poate ataca orice tip de inamic si rata de reincarcare intre atacuri este mica (dureaza mult timp sa se reincarce)
- **Machine Gun** (fig: 1) este turnul pe care il primim la inceputul jocului. Are viata mica, poate ataca orice tip de inamic, are raza medie, damage mic si rata de atac este mare (araca foarte des)
- **Electric Fence** (fig: 2) este turnul de protectie. Are viata foarte mare, poate ataca doar inamicii "melee", damage-ul este mediu si rata de reincarcare este mica. Scopul acestui turn este sa stea in calea inamicilor pentru a fi atacat de inamici, iar in timpul acela, celelalte turete pot distruge inamicii.
- **Vulkan** (fig: 3) Este tureta speciala impotriva inamicilor zburatori. Are damage mult dar poate ataca doar inamici zburatori, viata putina si raza de atac mare.
- **Machine Cannon** (fig: 4) este un turn care este foarte puternic impotriva inamicilor "melee". Are damage mare, rata de atac mica, viata medie, raza mica si prioritizeaza inamicii melee pe cat posibil.
- **Railgun** (fig: 5) este un turn care este foarte puternic impotriva inamicilor "ranged". Are damage mare, rata de atac mica, viata medie, raza mica si prioritizeaza inamicii ranged pe cat posibil.

- **Excavator** (fig: 6) este un tip special de turn deoarece nu poate ataca inamicii. Acesta aduna resurse in timpul luptei, resurse care pot fi convertite in bani de joc. Are viata multa, si pentru ca nu poate ataca, trebuie aparut de inamici.



Figure 1: Machine Gun

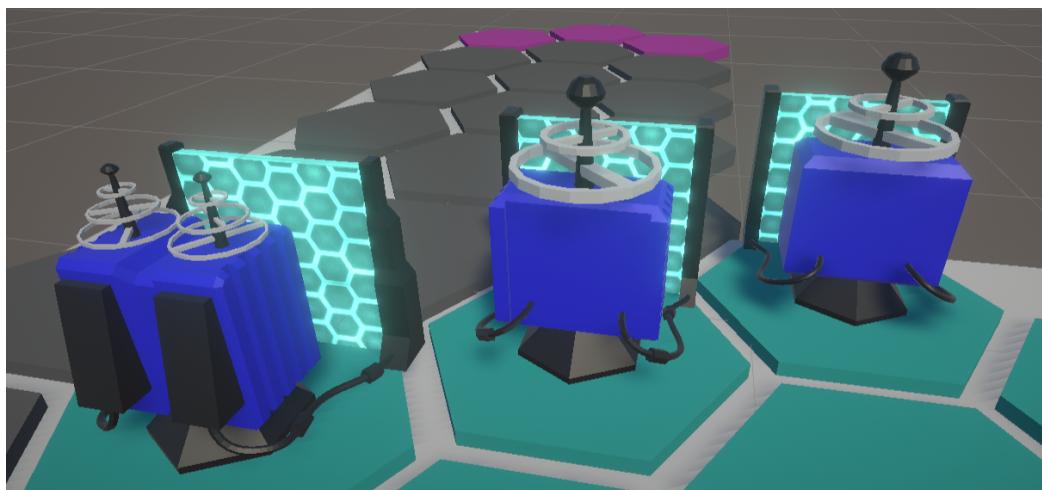


Figure 2: Electric Fence



Figure 3: Vulkan

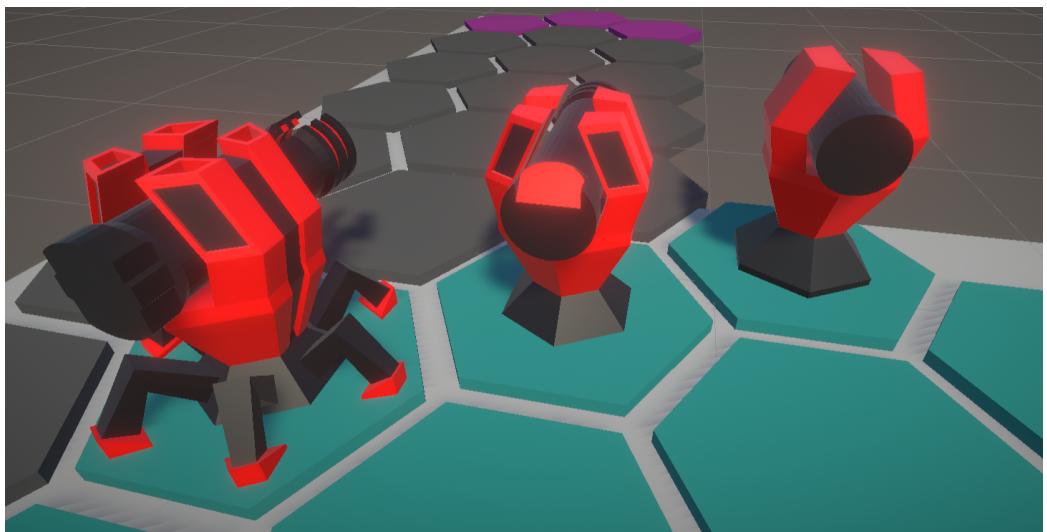


Figure 4: Machine Cannon

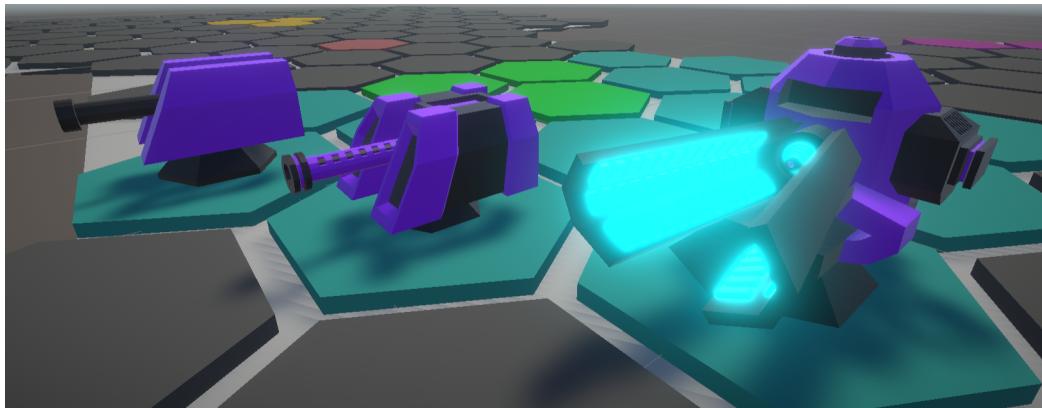


Figure 5: Railgun

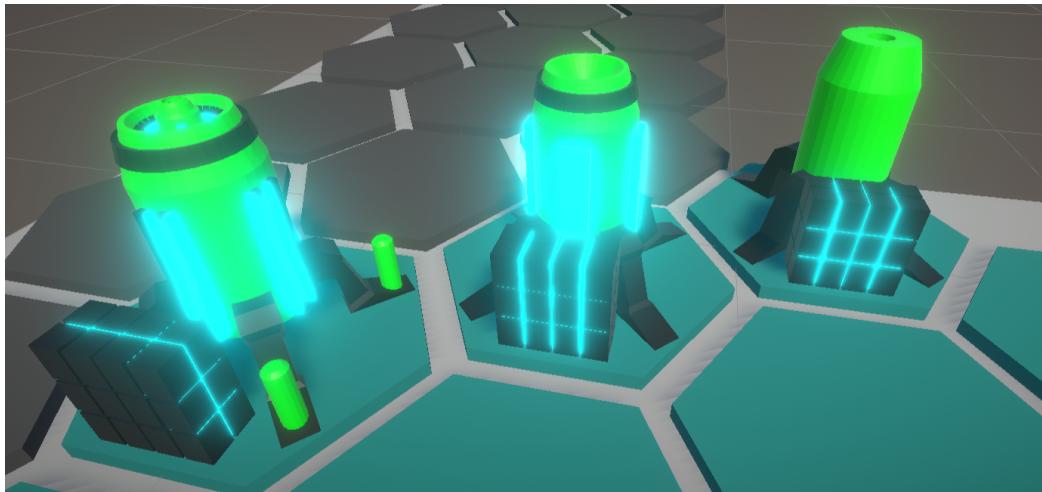


Figure 6: Excavator

3.4 Sistemele Jocului

3.4.1 Mapa

Pe mapa jocului vor fi generate automat obiecte hexagonale. Aceste obiecte pot fi de mai multe tipuri:

- Hexagoane goale (cele gri din (fig: 7)).
- Hexagoane pe care putem construi turnuri de atac. (cele albastre din (fig: 7))

- Hexagoane pe care putem construi turnul de extragere de resurse. (cele verzi din (fig: 7))
- Hexagonul pe care va incepe si se va afla comandanțul. Cand acesta se mișca, locul în care se mișca va deveni și el un astfel de hexagon. (cel roz din (fig: 7))
- Hexagoanele bazei. Baza va ocupa 3 hexagoane de acest tip. (cele galbene din (fig: 7))
- Hexagonul ocupat. Fiecare hexagon care este ocupat de o anumita structură devine un astfel de hexagon. (va avea culoarea roșie)

Inamicii își vor calcula drumul pe care merg în funcție de toate aceste tipuri de hexagoane. Din acest motiv este nevoie și de cel gol.

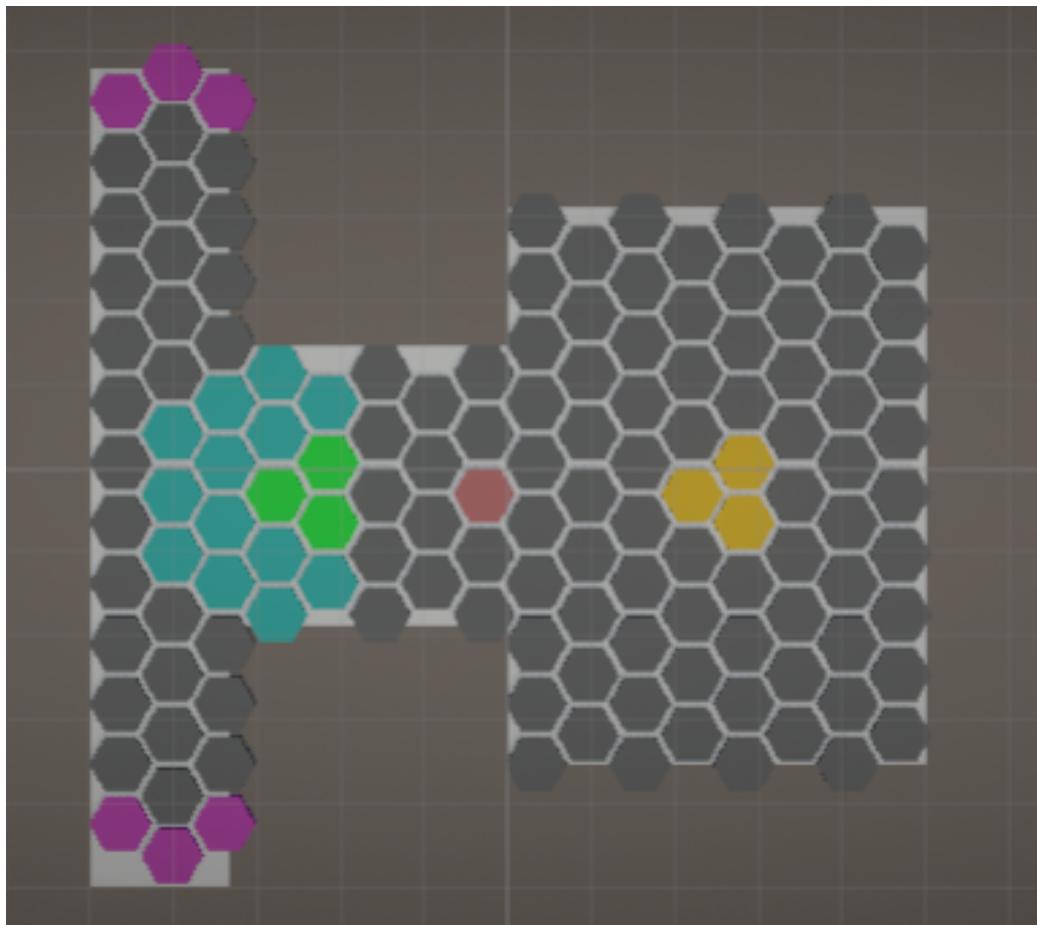


Figure 7: Hexagonal Grid

3.4.2 Sistemul Turnurilor

Turnurile pot fi plasate doar pe hexagoanele de constructie. Odata plasate, acestea nu pot fi mutate. Singurele metode de a goli acel hexagon este sa fie distrus de un inamic sau sa vindem tureta. Daca vindem o tureta, vom primi inapoi o parte din banii investiti. Acestea pot fi upgradate, dar acest sistem va fi deblocat la anumite stagii din joc. Toate turnurile au urmatoarele proprietati:

- Attack Speed
- Damage
- Health
- Range
- Attack type (specifica ce tip de inamici putem sa atacam)

Daca viata turetei ajunge la 0, va fi distrusa automat. Atata timp cat o tureta nu este distrusa, o putem repară, dar reparatiile vor fi executate intr-un anumit timp, iar in acest timp, tureta nu va putea sa atace.

3.4.3 Comandantul

Comandantul poate fi mutat pe orice hexagon care nu este deja ocupat. Dupa ce ai ales un hexagon, acesta va incepe sa se miste catre acesta. Cat timp este in miscare, nu poate ataca inamicii. Destinatia poate fi schimbată doar dupa ce a ajuns la destinatie. Are raza medie, rata de atac medie, damage mediu si poate ataca orice tip de inamic.

Acesta poate intra si in turete, imbunatatindu-le status-urile.

3.4.4 Tura de lupta (enemy wave)

Jocul va avea mai multe nivele, care pot fi selectate dintr-un meniu. Nivelele vor fi cat mai diferite posibil. Inamicii au mai multe puncte din care pot sa vina in functie de nivelul ales. Acestia vin in grupuri de inamici. Intre fiecare grup exista o perioada de repaus in care player-ul isi poate repară turnurile. In caz ca nu are nevoie de reparari, poate selecta sa sara peste perioada de repaus, astfel primind bani extra.

3.4.5 Nivelele jocului

Nivelul este terminat cand toti inamicii au fost omorati. Fiecare nivel va avea un scor de maxim 3 stele, care specifica cat de bine ne-am descurcat la acest nivel. Scorul este calculat in functie de mai multe proprietati, precum, viata ramasa a bazei la final, numarul de inamici omorati, numarul de turete ramase in viata (daca construim 12 si pierdem 2, acest scor nu va fi maxim). Fiecare stea castigata va da bani extra, bani care pot fi folositi in magazinul de upgrade-uri permanente, despre care vom discuta in scurt timp.

Playerul poate sa joace din nou nivalele, dar nu va mai primii atat de multi bani ca prima data. Aceasta va castiga bani extra de pe stelele castigate doar daca a primit mai multe stele decat turele anterioare.

3.4.6 Lock-on

Jucatorul poate sa apese pe inamici, astfel setand acel inamic drept o prioritate. Fiecare turn care poate ataca acest tip de inamic il va prioritiza fata de alti inamici. De indata ce este omorat, se pierde aceasta prioritate.

3.4.7 Magazinul de upgrade-uri permanente

Intre nivele putem sa cumparam piese de schimb pentru turnurile noastre pentru a le face mai puternice. Aceste piese se pot cumpara doar cu banii castigati de pe urma nivalelor. Upgrade-urile in functie de turn sunt urmatoarele:

- Machine gun: atac mai puternic, range mai mare, costuri mai mici.
- Electric fence: mai multa viata, atac mai puternic, costuri mai mici.
- Vulkan: atac mai puternic, range mai mare, atac mai rapid
- Machine cannon: atac mai puternic, range mai mare, atac mai rapid
- Railgun: atac mai puternic, range mai mare, atac mai rapid
- Excavator: da bani mai multi, timpul intre livrarile de bani mai scurt, cumpara upgrade-uri.

Fiecare upgrade va avea mai multe nivale, fiecare costand din ce in ce mai multi bani, dar vor avea un nivel maxim la care putem duce upgrade-urile.

3.4.8 Raid system

Cand un jucator alege acest mod, va putea sa aleaga din o serie de mape. Fiecare mapa are un monstru foarte puternic pe care trebuie sa il bata impreuna cu un alt jucator. Dupa ce a selectat mapa, va fi trimis la un ecran de asteptare, in care va sta pana cand se va gasi un alt jucator care a selectat acea mapa. Dupa ce a fost gasit un coechipier, vor incepe jocul. Undeva pe mapa se va afla un monstru imens si ei trebuie sa isi protejeze baza de acest monstru. Momentan tipurile de monstrii la care m-am gandit sunt urmatoarele:

- **Fire Demon** este un monstru din lava care din cand in cand loveste turetele si baza, provocand o tona de daune. Monstrul trebuie omorat cat de repede posibil ca sa nu distruga baza inainte.
- **Necromancer** este un monstru care cheama alti monstrii pe mapa. Puterea lui sta in spatele intregii armate ale lui. Daca playerii sunt complesiti de aceasta armata vor pierde nivelul. Trebuie sa aiba un echilibru bun intre defensiva si ofensiva.
- **Armadillo** este un care ataca periodic baza. Da mai putin damage decat "Fire Demon", dar are si un tip de atac in care se face ghem si incepe sa se roteasca. Daca playerii nu ii dau suficient de mult damage in timpul ala sau nu fac anumite actiuni specifice (ex: sa mute comandanțul intr-un anumit loc sau sa construiasca o tona de turete in calea lui), el va ataca baza, dand foarte mult damage.

Fiecare player poate plasa turete oriunde. Playerii pot plasa turete unul peste tureta celuilalt, astfel activand un efect special. Acestia pot sa isi upgradeze doar turetele construite de ei. Amandoii playerii au caracterele lor comandanți, dar acestia nu pot intra in aceeasi tureta in acelasi timp. Pentru sistemul de lock-on, doar turetele player-ului care a setat lock-on-ul va ataca inamicul prioritizat.

3.5 Analiza proiectului din punctul de vedere al consumatorilor

3.5.1 Dificultatea jocului

Jocul va fi relativ dificil. Trebuie sa te gandesti unde sa construiesti anumite turete pentru a folosi resursele cat mai bine. La inceput jocul va fi usor, dar pe urma va trebui sa te gandesti foarte bine unde sa plasezi anumite

turete, altfel vei pierde nivelul. Trebuie sa se gandesti si daca poti rezista sa sari peste perioada de repaus intre grupurile de inamici.

Sistemul de raid are un nivel de dificultate cu mult mai ridicat, pentru ca trebuie sa te si coordonezi cu un alt jucator.

3.5.2 Elemente de dependenta

In caz ca nu putem depasii un anumit nivel, putem juca nivelele anterioare si sa ne asiguram ca luam 3 stele la fiecare nivel. Aceasta metoda de a obtine 3 stele la fiecare nivel poate fi unul din motivele care ii determina pe jucatori sa continue sa joace jocul. Un alt factor ar putea fi sistemul de upgrade-uri permanente.

3.5.3 Grupele de varsta vizate

Acest joc nu are limitari de varsta, poate fi jucat de oricine, dar va fi apreciat cel mai mult de copii si de adolescentii care cauta provocari in jocurile de strategie. Jocul este in principiu facut pentru "casual gamers".

4 Implementare proiect

4.1 Scena de lupta

4.1.1 Initializarea scripturilor

In unity scripturile au cateva metode care ajuta la initializare.

Awake este metoda care este apelata la inceputul programului si inainte de restul metodelor.

Start este metoda care este apelata dupa ce s-au apelat metodele Awake de la toate scripturile.

Update este apelat la fiecare frame al jocului, aici intervine o mare parte din logica jocurilor. Apelarea update-ului incepe dupa ce toate metodele "Start" au fost apelate.

Pentru "Start" si "Awake", unity decide automat in ce ordine sa le apeleze, fara ca noi sa putem controla acest lucru. In multe proiecte mai complexe se ajunge in momentul cand pentru initializarea anumitor sisteme este necesar ca alte sisteme sa fie deja initializate. Cum nu putem controla ordinea de initializare, se ajunge in punctul in care putem primi erori random din cauza initializarii haotice.

Pentru a combate asta, m-am gandit la un sistem care va permite controlarea initializarii tuturor procese cu usurinta.

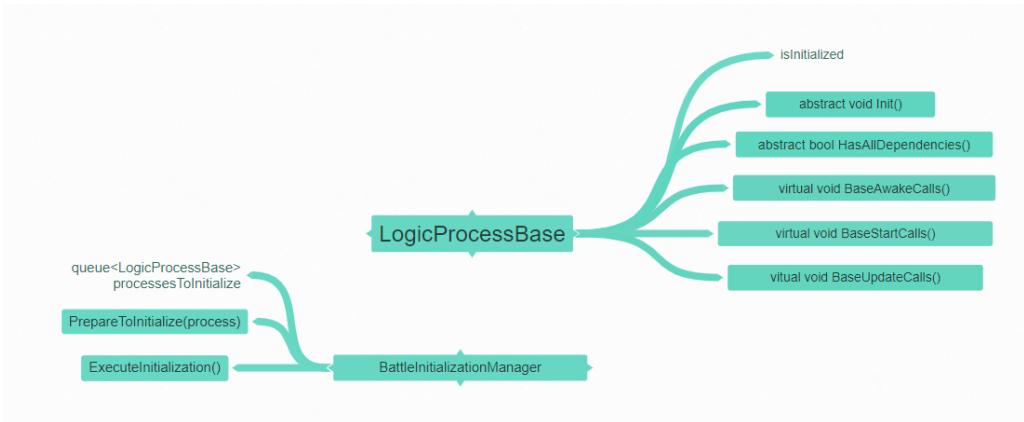


Figure 8: Initializarea scripturilor

Am creat clasa LogicProcessBase, pe care, toate scripturile care au nevoie de o initializare mai organizata, trebuie sa o mosteneasca si care contine:

- **isInitialized** care specifica daca scriptul curent a fost initializat sau nu inca
- **Init()**. In aceasta metoda trebuie sa scriem tot codul de initializare a clasei care mosteneste.
- **HasAllDependencied()**. Este o metoda care returneaza un boolean. In aceasta clasa trebuie sa punem toate dependintele de care are nevoie clasa care mosteneste.
- **BaseAwakeCalls()**. In aceasta metoda apelam ”PrepareToInitialize()” din clasa ”BattleInitializationManager”, despre care vom vorbi in scurt timp.
- **BaseStartCalls()** si **BaseUpdateCalls()** sunt metode in care trebuie sa scriem tot codul care normal ar fi venit in Start/Update. In unity, nu putem defini Start/Update drept virtual si sa modificam functiile lor pe urma, din acest motiv tot codul din clasele dintr-o ierarhie mai complexa trebuie sa fie scris in aceste metode, iar in clasa cel mai jos din ierarhie trebuie sa le apelam in Awake/Start/Update.

BattleInitializationManager este a doua clasa de care avem nevoie. Ea este responsabila pentru a executa initializarea propriuza a proceselor.

PrepareToInitialize() este apelata de fiecare proces in metoda lor de awake, iar aceasta metoda va adauga procesul intr-o coada.

ExecuteInitialization() parurge toate procesele din coada si verifica daca acestea pot fi initializate, apeland ”HasAllDependencies()”. Daca procesul curent poate fi initializat, atunci ii apeleaza metoda ”Init()” si seteaza ”isInitialized = true”. Daca nu il poate initializa, il adauga la finalul cozii pentru a fi initializat la final. Acest proces se repeta pana cand coada este goala sau s-a parcurs o data coada cap-coada si nu s-a initializat nici un proces. In acest caz inseamna ca aveau dependinte circulare si oprim procesul de initializare, afisand un mesaj de eroare. Aceasta metoda trebuie apelata in Awake, Start si Update, deoarece, pe parcurs pot aparea noi procese care trebuie initializate, iar programul trebuie sa poata sa detecteze astfel de cazuri.

4.1.2 Procesarea comenziilor venite de la utilizator

4.1.3 Grid system

4.1.4 Sistemul de navigare

Caracterele jocului trebuie sa stie pe unde pot sa meargă ca sa ajunga la o anumita destinație. În acest scop a trebuit să aleg un algoritm care să îmi gasească cel mai scurt drum posibil. După o investigare amanuntita am ales să folosesc algoritmul Floyd-Warshall.

”Consideram reteaua orientata $G = (N, A, b)$ reprezentata prin matricea valoare adiacenta $B = (b_{ij}), i, j \in N$ cu

$$b_{ij} = \begin{cases} b(i, j) & \text{daca } i \neq j \text{ si } (i, j) \in A; \\ 0 & \text{daca } i = j; \\ \infty & \text{daca } i \neq j \text{ si } (i, j) \notin A. \end{cases}$$

Algoritmul Floyd-Warshall determină matricea distanțelor $D = (d_{ij}), i, j \in N$ și matricea predecesor $P = (p_{ij}), i, j \in N$.” [3]

```

function FLOYD-WARSHALL
    for i  $\leftarrow$  1 to n do
        for j  $\leftarrow$  1 to n do
             $d_{ij} \leftarrow b_{ij};$ 
            if i  $\neq$  j and  $d_{ij} < \infty$  then
                 $p_{ij} = i;$ 
            else
                 $p_{ij} = 0;$ 
            end if
        end for
    end for

```

```

for k  $\leftarrow$  1 to n do
    for i  $\leftarrow$  1 to n do
        for j  $\leftarrow$  1 to n do
            if  $d_{ik} + d_{kj} < d_{ij}$  then
                 $d_{ij} = d_{ik} + d_{kj};$ 
                 $p_{ij} = p_{kj};$ 
            end if
        end for
    end for
end function

function RECONSTRUIRE DRUM
    k = n;
     $x_k = j$ 
    while  $x_k \neq i$  do
         $x_{k-1} = p_{ix_k};$ 
        k = k - 1
    end while
end function

```

Drumul minim este $D_{ijp} = (x_k, x_{k+1}, \dots, x_{n-1}, x_n) = (i, x_{k+1}, \dots, x_{n-1}, j)$

O alternativa pe care am considerat-o pentru algoritmul de navigare a fost algoritmul lui Dijkstra. Acesta este de asemenea un algoritm care gaseste cel mai scurt drum intre 2 noduri si se bazeaza pe relatiile urmatoare:

$$\begin{aligned}
 d(s, x_h) &= \sum_{i=1}^h b(x_{i-1}, x_i). \\
 d(x_{j+1}) &\leq d(x_j) + b(x_j, x_{j+1}) = d(s, x_j) + b(x_j, x_{j+1}) = d(s, x_{j+1}) \leq d(s, z). \\
 d(y) &= \min\{d'(y), \min\{d'(x) + b(x, y) | x \in V^-(y)\}\}.
 \end{aligned}$$

In cele din urma am ales Floyd-Warshall, deoarece, chiar daca are complexitatea mai mare, va fi rulat o singura data in tot programul, iar reconstruirea drumului se intampla aproape instantaneu. Mapa nu se va schimba pe parcursul jocului, iar din acest motiv am considerat ca e o optiune mai buna decat alti algoritmi precum Dijkstra sau A*.

4.1.5 Ierarhia inamicilor

4.1.6 Ierarhia turnurilor defensive

4.2 Selectarea nivelelor

4.3 Magazinul de upgrade-uri permanente

4.4 Sistemul co-op

5 Concluzii

References

- [1] Mike McShaffry, David Graham.
Game Coding Complete Fourth Edition. 2012
- [2] Jesse Schell.
The Art of Game Design: A Book of Lenses 1st Edition. 2008
- [3] Eleonor Ciurea, Laura Ciupala.
Algoritmi: Introducere in algoritmica fluxurilor in retele. 2006