**Dungeon Trader**

MANEA Mihai-Alexandru si

SIRGHIE-CIARCIANI David

Departamentul de Informatica Economica

Universitatea Romano-Americana

*Abstract.*

This is project is made by two students that really like dungeon crawlers. We made this game by combining the concepts of dungeon crawler type games with concepts from the stock market / cryptocurrency market. The game was made by using the Unity Real-Time Development Platform and the C# coding language. The game levels and maps are procedurally generated. In continuation we will present you how our game works and some interesting algorithms that we wrote for this project.

*Cuvinte Cheie. Joc . Unity . C# . Dungeon . Trader*

**Introducere.**

Aceasta lucrare este un joc facut de la zero folosind Unity Real-Time Development Platfor si limbajul de programare C#. Am ales sa facem un joc deoarece ne plac jocurile de tip „Dungeon Crawler” si „Rogue Like” , iar acest joc facut de noi include multe concepte din aceste tipuri de jocuri dar am introdus si o componenta unde norocul este un factor important.

In acest proiect am invatat cum sunt facute multe din jocurile noastre preferate, cat de complicata este imbinarea front-end cu back-end si folosirea GameEngine-ului de la Unity. Totodata ne-a adus bucurie atunci cand munca noastra dadea roade si le vedeam direct in fata noastra pe ecran , munca in echipa si coordonarea au fost de nelipsit si pentru prima data putem sa spunem ca am facut parte dintr-o echipa de development.

In continuare, vom prezenta aplicatia vizuala, conceptele si algoritmii pe care i-am folosit, cum am gandit jocul si cum am procedat in implementarea lor.

**Explicarea Conceptelor.**

Dungeon Crawler este un tip de scenariu in jocurile RPG(Role playing games) in care eroul navigheaza un mediu inconjurator de tip labirint in timp ce se lupta cu diferiti monstrii , ocolind capcanele, rezolvand puzzle-uri si colectand diferite comori.

Roguelike este o subdiviziune a jocurilor RPG (Role Playing Games) care traditional era caracterizata ca un Dungeon Crawler dar cu nivele generate procedural, jocul fiind „turn-based”(bazat pe ture), miscarea are loc pe un grid(tabla de joc) iar moartea este

permanenta. Aceste trasaturi ale genului Roguelike ofera o latura misterioasa unui joc, niciodata nu te astepti la ce o sa urmeze si trebuie sa te pregatesti pentru ce este mai rau la fiecare pas, miscarea pe tabela prin ture este simpla dar in acelasi timp lasa loc de o gandire strategica si logica mult mai amplificata decat in alte jocuri. Majoritatea jocurilor Roguelike sunt influentate foarte mult de narativul „fantezie”, o subspecie a fantasticului, aceste lucruri arata ca acest gen de jocuri video este influentat de jocurile de comunitate stil „Table Top”.

Noi am decis sa combinam aceste concepte cu abilitatea de a investi banii stransi pe parcursul aventurii si din profitul obtinut sa cumparam imbunatatiri fara de care aventura nu poate avea loc, dificultatea creste destul de repede si fara investitii nu avem cum sa facem fata tuturor inamicilor.

**Elementele Proiectului**

Am despartit tot proiectul in cinci „manageri”, interfata grafica si sistemul de investitie si cel de cumparare al imbunatatirilor.

Managerul tablei, acest manager este un script scris in C# care genereaza toata tabla de joc folosind un dictionar de vectori tip „Vector2” care contine coordonatele fiecarei casute si componenta de joc „Tile” care este oferita de Unity. Aceasta clasa Tile are si un script separat care ii ofera diferite proprietati, precum proprietatea playerOnIt sau enemyOnIt ce ne vor ajuta mai tarziu pentru derularea jocului propriu-zis, sau culorile si highlight-urile folosite pentru a colora cum dorim tile-urile generate si pentru a arata highlight-uri pentru miscare, atac, etc. Pentru a pastra partea de generare procedurala , folosim metoda „Range” din clasa „Random” pentru a genera doua valori aleatorii. Prima valoare este pentru latimea tablei si cealalta este pentru inaltimea tablei.

Dupa ce am generat parametrii tablei de joc incepem sa parcurgem cu doua for-uri ca pe o matrice, generand fiecare casuta din tabela noastra si schimbandu-le culoarea alternativ pentru a oferi un contrast crescut asupra fiecarei casute. Dupa ce am generat toata tabla de joc ne apucam si oferim componentei „GameCamera” pozitia de unde ne va oferi perspectiva perfecta asupra tablei. La final generam coordonatele unde vom plasa pe tabla jucatorul fara de care aventura noastra nu poate sa aiba loc.

In managerul tablei avem si functiile pentru identificarea coordonatelor de care vom avea nevoie in restul scripturilor de manageri. Avem functii pentru obtinerea coordonatelor unde am plasat jucatorul, functie pentru obtinerea coordonatelor unde am plasat inamicul, o functie pentru a returna casuta de la pozitia pe care o vrem noi, o functie pentru a obtine pozitia unei casute, o functie care cauta coordonatele jucatorului. Tot aici avem si functiile care ne ofera un ajutor vizual asupra actiunilor pe care le putem face in joc, avem o functie pentru a marca casutele unde ne putem muta, una care marcheaza inamicii, o functie care marcheaza casutele unde putem sa atacam si o ultima functie care sterge toate ajutoarele vizuale de unde nu ar trebui sa fie.

Managerul jucatorului se ocupa de generarea jucatorului prin intermediul clasei „Player”, prima actiune este aceea de a obtine coordonatele casutei care a fost aleasa de catre managerul tablei pentru plasarea jucatorului. Generam un vector de tip „Vector3” unde salvam coordonatele in spatiul Unity cu valoarea „-0.01f” pentru axa Z pentru a aparea pozitionat deasupra tablei de joc si nu in interiorul ei si oferim informatia casutei ca jucatorul este acolo. Imediat dupa aceste actiuni incepem sa generam inamicii prin intermediul managerului de joc.

Restul componentelor din managerul jucatorului sunt functii pentru setarea casutelor unde putem sa atacam, o functie pentru mutarea jucatorului in care obtinem pozitia lui actuala de la managerul tablei, setam existenta jucatorului in aceea casuta ca falsa, curatam toate indicatoarele vizuale, obtinem coordonatele unde am ales sa ne mutam si setam prezenta jucatorului iar ultima actiune este aceea de a incheia tura urmand tura inamicilor. Functia „PassPlayer” este folosita pentru a oferi acces celorlalte scripturi la componenta „player” si ultima functie este folosita pentru a distruge obiectul „jucator”.

Cel mai important manager este managerul jocului, prin intermediul lui controlam „stadiile” jocului si fara de care nu am putea sa coordonam restul managerilor. Acest script foloseste un switch in care apelam anumite functii sau setam anumite valori totul pentru a coordona rolurile fiecarui „manager”.

In interiorul switch-ului gasim mai multe cazuri denumite si „GameState” sau stadiile jocului. Am utilizat stadiul de „Menu” pentru a detecta cand suntem in stadiul initial in meniu, folosim stadiul de „Setup” pentru a seta numarul de camere la 0 si nivelul la 0 , valori initiale si sarim la urmatorul stadiu. „NewGrid” , unde generam o tabela noua. In cadrul functiei de generare a tabelei o sa sarim la stadiul „SpawnPlayer” in care din ce am explicat la managerul jucatorului o sa fie facuta o saritura la stadiul „SpawnEnemies” unde se utilizeaza managerul inamicilor si se seteaza tura la 1.

Mai avem cateva stadii mai minore care ofera componenta de „turn based”, stadiul „playerTurn” care incepe tura jucatorului , stadiul „EnemyTurn” care incepe tura inamicilor, Stadiul „Lose” care acopera ecranul cu un overlay rosu si ultimul stadiu cu numele de „Load” apeleaza o functie pe care o voi detalia putin mai tarziu in prezentare.

Functiile pe care le gasim in acest manager sunt o functie pentru a pasa numarul turii curente, una pentru a oferi informatia ca jucatorul a terminat tura, o functie care face tura jucatorului in care crestem numarul turii, cerem managerului jucatorului sa ne ofere obiectul jucator, il initializam cu valorile de care avem nevoie (viata, atac, etc.) folosind o functie, verificam daca viata jucatorului este mai mica sau egala cu 0 si in caz pozitiv distrugem obiectul jucator si schimbam stadiul jocului cu „Lose”. Daca jucatorul nostru este inca in viata incepem sa setam toate indicatoarele vizuale. Functia urmatoare permite verificarea stadiului jocului, daca nu exista inamici schimbam stadiul jocului la „Load” generand un nou nivel. In cazul in care inca avem inamici , trecem la tura jucatorului.

Functia „DoLoad” care este apelata in stadiul „Load” ne permite sa distrugem obiectul „Player”, sa stergem tabela, crestem numarul camerelor prin care am trecut, crestem nivelul la fiecare 5 camere si generam o tabela noua.

Managerul de nivel este cel mai simplu dintre scripturile managerilor, acest manager contine doua functii , una prin care pasam nivelul curent si una prin care cerem nivelul curent.

Ultimul manager si cel mai complicat este managerul inamicilor. Am decis pentru inceput sa avem trei tipuri de inamici cu dorinta de a adauga mai multi pe parcurs. Incepem prin a declara clase „Enemy” pentru fiecare tip de inamic, fiecarui inamic o sa ii corespunda un punctaj si mai declaram un dictionar cu coordonatele fiecarui inamic.

Prima functie din managerul inamicilor este functia de plasare a inamicilor pe tabela de joc. Se creeaza dictionarul de inamici, creeam si un vector de frecventa pentru numarul fiecarui tip de inamici, cerem numarul nivelului curent de la managerul nivelului, setam punctajul alocat nivelului si utilizam acest punctaj pentru a determina numarul de inamici si dificultatea nivelului prin introducerea unor tipuri noi de inamici. Acest lucru se face printr-o structura repetitiva care ruleaza atata timp cat avem punctaj ramas sau cat timp avem unde sa pozitionam inamicii pe tabela. Urmatorul pas este de a genera un numar aleatoriu dintre 0,1 si 2, verificam daca punctajul inamicului careia ii corespunde valoarea aleatorie se incadreaza in ce a ramas din punctajul nivelului si daca da intram in switch unde setam tipul inamicului respectiv. Cu aceasta informatie trecem la pozitionarea propriu-zisa a inamicului pe tabela, acest lucru se face la fel ca la jucator numai ca in loc sa setam un parametru care sa determine existenta unui jucator pe casuta, setam un parametru pentru a determina existenta unui inamic pe casuta. Dupa terminarea pozitionarii inamicilor pe tabela trecem la tura jucatorului.

Mai avem in managerul inamicilor functii pentru obtinerea casutei unde se afla un inamic, o functie pentru a pozitiona pe o casuta un inamic. Dupa aceste functii avem o functie care este utilizata in atacul jucatorului asupra unui inamic, aceasta functie obtine coordonatele casutei alese, verifica daca se afla inamicul la casuta si daca da scade din viata inamicului valoarea atacului detinuta de jucator si in cazul in care viata inamicului scade sub 1 o sa oferim jucatorului o suma de bani corespunzatoare punctelor alocate inamicului inmultite cu doi si stergem orice urma de inamic de pe tabela. In final stergem semnele vizuale de pe tabela si terminam tura jucatorului.

Urmatoarea functie folosita este o functie care ia dictionarul in care am salvat inamicii plasati pe tabela in ordinea in care i-am plasat si verifica tipul lor , in functie de tipul lor apelam functia corespunzatoare tipului de miscare pe care inamicul o poate face.

Primul tip de inamic este „nebunul”, el stie sa faca un singur lucru, sa mearga direct la jucator si sa il atace dar se poate misca cate o casuta iar punctajul alocat acestui inamic este cel mai mic, doua puncte. Pentru a determina miscarea nebunului o sa declaram o un vector simplu in care salvam scorul miscarilor posibile, nebunul se misca o singura casuta deci exista opt miscari posibile pentru care calculam scorul. Obtinem pozitia nebunului si pozitia jucatorului si introducem pozitiile casutelor unde putem sa facem actiunea in doi vectori, un vector pentru coordonatele X si un vector pentru coordonatele Y. Urmatorul pas este sa calculam scorul fiecarei miscari posibile, am folosit pur si simplu suma diferentelor de coordonate dintre nebun si jucator. Dupa determinarea scorului incepem sa cautam cel mai mic scor care corespunde si cu cea mai buna miscare si verificam daca la pozitia unde urmeaza sa ne mutam putem sa atacam sau sa ne miscam. Daca putem sa atacam o sa atacam jucatorul scazand valoarea atacului nebunului din viata jucatorului. Daca putem sa ne miscam in acea casuta atunci ne miscam prin mutarea inamicului la coordonatele noi, setarea prezentei lui pe casuta de la acele coordonate si stergerea prezentei inamicului de pe casuta veche.

Al doilea tip de inamic este „Charger-ul” , acest inamic isi incarca atacul in turele impare si alearga pana la cea mai apropiata pozitie de jucator indiferent de distanta. Incepem la fel ca la nebun dar totul se schimba dupa ce obtinem informatiile necesare. Verificam care din variante este mai buna , varianta de a merge in stanga sau dreapta si varianta de a merge sus sau jos printr-o scadere de coordonate dintre inamic si jucator. Dupa ce am facut aceasta verificare mergem pe cazurile favorabile unde incepem sa testam daca putem sa ne mutam catre jucator, in cazul in care ajungem la o casuta care nu este declarata ne oprim dar daca ajungem direct la jucator o sa setam variabila care ne spune ca putem sa atacam jucatorul. Dupa ce verificam toate variantele , atacam jucatorul daca ne intersectam cu el altfel ne mutam la pozitia cea mai favorabila pe care am determinat-o anterior.

Ultimele functii din managerul inamicilor ne permite sa cerem daca mai exista inamici pe tabela si sa distrugem obiectul inamic.

Interfata grafica este creeata folosind trei obiecte „Canvas” fiecare fiind legat de o camera separata pentru a putea schimba intre ele mai usor fara a creea conflicte. Fiecare canvas are in componenta panouri, butoane si text. Toate aceste elemente interactioneaza intre ele pentru a oferi o interfata grafica pentru utilizatori.

Am impartit interfata grafica in trei componente: Prima componenta este interfata meniului principal de unde jucatorul poate sa inchida jocul sau sa porneasca un joc nou la fiecare apasare de buton.

A doua componenta este interfata jocului unde oferim informatii pentru utilizatori, in aceasta interfata viata, banii detinuti de jucator si banii investiti, butoanele pentru meniu, pentru panoul cu informatiile despre jucator, pentru panoul de investitii unde jucatorul poate sa investeasca si ultimul buton este butonul care schimba camera pe canvasul folosit pentru magazin. Panoul de investitii este o componenta direct construita in interfata principala a jocului, printr-o apasare de buton deschidem panoul de investitii si avem accesul la cel de-al doilea concept al jocului nostru. Fara investitii in joc nu poti inainta, dificultatea creste exponential iar jucatorul este nevoit sa isi administreze banii pentru a face fata dificultatii prin cumpararea de imbunatatiri.

Panoul de investitii este administrat printr-un script care se ocupa de investirea banilor si scoaterea lor din „market” astfel incat sa fie folositi in magazin. Investirea banilor se face printr-un „Input Field” si la apasarea butonului de investitie convertim stringul intr-o variabila de tip intreg si verificam daca aceasta valoare este mai mare decat banii disponibili , in cazul pozitiv stergem continutul campului text iar in cazul negativ o sa mutam banii in variabila folosita pentru investirea banilor. Scoaterea banilor din „market” se face prin introducerea valorii dorite in „Input Field”-ul respectiv acestei actiuni si la apasarea butonului convertim stringul intr-o valoare intreaga si verificam daca depaseste banii disponibili in contul de investitii, daca este afirmativ atunci stergem continutul campului text si daca este negativ atunci transferam banii in variabila contului curent de unde putem sa ii folosim.

Panoul de investitii functioneaza utilizand sistemul bazat pe ture de joc si in fiecare tura se apeleaza functia „MarketTurnTick” care ruleaza doar daca avem bani investiti si genereaza o valoare aleatorie de la unu la o suta si una aleatorie de la zero la cincizeci. Folosim a doua valoare aleatorie ca procent din banii investiti si calculam rezultatul , dupa actualizam textul care va afisa istoricul ultimelor patru ture de market si verificam daca prima valoare aleatorie a generat un castig sau o pierdere. Daca valoarea generata este mai mare ca „MarketOddsLimit” –(2 \* „Luck”) atunci este un castig daca este sub atunci este o pierdere , „MarketOddsLimit” a fost initializat cu 50 iar „Luck” cu zero dar poate sa fie crescut prin achizitii in magazin. In caz de castig o sa adaugam in contul de investitii valoarea castigului calculata anterior pe baza celei de a doua valori aleatorii. In caz de pierdere o sa scadem banii din contul de investitii.

A treia componenta este magazinul de unde putem sa cumparam imbunatatiri pentru jucator. Imbunatatirile din magazin sunt: Viata, atacul, distanta de atac, norocul si recuperarea vietii. Fiecare achizitie a unei imbunatatiri o sa creasca costul imbunatatirii respective.

Daca utilizam magazinul si panoul de investitii , tura noastra se va termina imediat ce inchidem interfata respectiva si va incepe tura inamicilor. Daca distrugem toti inamicii o sa avansam la camera urmatoare , la un anumit numar de camere crestem nivelul si odata cu nivelul o sa creasca si atacul inamicilor, viata inamicilor, punctajul disponibil pe camera pentru plasarea inamicilor.

**Incheiere.**

In incheiere vom spune ca am invatat multe pe parcursul proiectului, ne-a facut placere sa lucram la proiect chiar daca uneori am fost frustrati sau nervosi ca nu mergea o anumita linie de cod sau mai multe linii, pana la urma asta este placerea de a fii programator. Dezvoltarea unui joc initial parea o treaba usoara dar cu cat doream sa adaugam componente noi si mai complexe cu atat incepeam sa realizam ca mai avem mult de invatat, am descoperit clase noi, metode noi si avand in fata un prototip functional de joc pe care l-am facut noi de la zero ne face sa ne simtim plini de viata chiar si dupa multe zile in care ne-am concentrat pe acest proiect.