UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

**Facultatea de Informatică**

****

Lucrare de licenţă

**BattleOfTitans**

Propusă de

***Cehan Dan Ștefan***

**Sesiunea:**Iulie, 2017

Coordonator Ştiinţific:

***Asistent, dr. Vasile Alaiba***

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

**Facultatea de Informatică**

**BattleOfTitans**

z

***Cehan Dan Ștefan***

**Sesiunea:**Iulie, 2017

Coordonator Ştiinţific:

***Asistent, dr. Vasile Alaiba***

**DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA**

**DREPTURILOR DE AUTOR**

Prin prezenta declar că Lucrarea de licență cu titlul “BattleOfTitans” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învățământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

−toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă,

sunt scrise între ghilimele şi deţin referinţa precisă a sursei;

−reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa

precisă;

−codul sursă, imagini etc. preluate din proiecte open source sau alte surse sunt utilizate

cu respectarea drepturilor de autor şi deţin referinţe precise;

−rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi,

Absolvent Cehan Dan Ștefan

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

**DECLARAŢIE DE CONSIMŢĂMÂNT**

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „BattleOfTitans”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică. De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi,

Absolvent Cehan Dan Ștefan

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (semnătura în original)

Cuprins

[1 Introducere 6](#_Toc484884973)

[1.1 Motivație 6](#_Toc484884974)

[1.2 Context 6](#_Toc484884975)

[1.3 Cerințe funcționale 10](#_Toc484884976)

[1.4 Abordare tehnică 10](#_Toc484884977)

[2 Contribuție 12](#_Toc484884978)

[3 Proiectare 13](#_Toc484884979)

[3.1 Arhitectura soluției 13](#_Toc484884980)

[3.2 Modelarea datelor 21](#_Toc484884981)

[3.2.1 Modelarea datelor pe partea de client 22](#_Toc484884982)

[3.2.2 Modelarea datelor pe partea de server 23](#_Toc484884983)

[3.3 Protocoale de comunicare client – server 24](#_Toc484884984)

[3.4 Interfața cu utilizatorul 26](#_Toc484884985)

[4 Implementare 29](#_Toc484884986)

[4.1 Chat 29](#_Toc484884987)

[4.2 Sistemul de camere 31](#_Toc484884988)

[4.3 Algoritmul A\* Pathfinding 32](#_Toc484884989)

[4.3.1 Introducere 32](#_Toc484884990)

[4.3.2 Binary Heap 35](#_Toc484884991)

[4.3.3 Prezentare algoritm 37](#_Toc484884992)

# Introducere

## Motivație

Consider că, pentru a se relaxa la un moment dat, fiecare dintre noi a experimentat un joc video pe calculator. Multe jocuri existente pe această imensă platformă pot accentua stresul deoarece pot exista anumite impedimente pe parcusul explorării acestora. Deși au farmecul lor, de cele mai multe ori lumea dorește o variantă mai simplă, având o durată mai scurtă de timp și care oferă o satisfacție mai mare încă de la început.

Această lucrare de licență are ca scop oferirea unui serviciu online de entertainment sub forma unui joc video de strategie în maniera 3D, de tip multiplayer online battle arena (MOBA).

Aplicația va stimula capacitatea jucătorului de a lua decizii rapide la fiecare pas, fără posibilitatea de a planifica totul din timp. Acest lucru va conduce la încercarea mai multor strategii în timpul jocului și adaptarea lor la fiecare pas. Ea va fi disponibilă doar pe Microsoft Windows.

## Context

Multiplayer online battle arena (MOBA) este un sub gen al jocurilor video de [strategie](https://ro.wikipedia.org/wiki/Strategie_%C3%AEn_timp_real) în timp real, în care jucătorul controlează un singur caracter, fiind plasat într-o echipă din cele două. Obiectivul este distrugerea structurilor inamice cu ajutorul\* unităților generate periodic, unitați ce au un drum prestabilit.

Jucătorii au în general abilități și avantaje variabile care se îmbunătățesc odată cu trecerea timpului și care contribuie la strategia echipei per ansamblu. Pe parcursul jocului, ei vor câștiga puncte de experiență și skilluri noi, gold, echipament și vor putea distruge anumiți monstri din junglă. MOBA este o fuziune între un joc de acțiune și un joc de strategie în timp real, unde jucătorii nu pot construii clădiri sau unități.

Deoarece am dorit nu doar sa mă joc, ci și să implementez un joc din această categorie, am hotârat sa dezvolt aplicația BattleOfTitans. Acesta este un joc în manieră 3D, fiind inspirat din jocurile LeagueOfLegends&Dota2, unde utilizatorul va fi în competiție cu alți protagoniști. Competiția se va realiza in modul 1vs1, fiecare utilizator având control asupra unei unități erou, unitate ce va dispune de anumite abilități magice. Fiecare jucător va fi poziționat intr-o bază ce va avea nevoia de protecție. De asemenea fiecare jucător este ajutat de anumite unități ce vor participa impreună la distrugerea turnurilor de protecție ale oponentului. Cel care distruge primul construcția finală a adversarului va câștiga.

**Aeon of Strife**

În 1998, un moderator numit Aeon644 a creat Aeon of Strife, o hartă personalizată creată de un fan pentru jocul real-time Starcraft al celor de la Blizzard . Pe acestă mapă, jucătorii controlau un singur caracter și se luptau cu o echipă adversă, ale carei unități erau controlate de calculator si dirijate pe trei benzi. Benzile conectau bazele celor două echipe. Obiectivul era distrugerea bazei celeilalte echipe.

Pornind de la baza creată de Aeon of Strife pentru genul MOBA putem obeserva anumite diferențe notabile față de un adevărat MOBA de azi. În primul rând, echipele aveau câte patru jucători în loc de cinci. De asemenea, Aeon of Strife nu a fost un joc în maniera competitivă, echipă de eroi controlați de jucători se confruntau cu o echipă de personaje controlate de calculator. Eroii nu se dezvoltau în timp ce jocul progresa. De asemenea nu exista jungla suplimentară și drumurilele auxiliare între cele trei benzi.

**Defense of the Ancients**

În anul 2002 cei de la Blizzard au eliberat următorul lor joc real-time, Warcraft III. La fel ca și Starcraft, acesta a venit cu anumite unelte care permiteau utilizatorilor să creeze mape și scenarii personalizate. În 2003, un editor de harți numit Eul a creat, fiind inspirat din Aeon of Strife, un modul numit Defense of the Ancients(DOTA). Imediat dupa ceilalți jucători au creat versiunile lor proprii, fiecare adăugând proprii lor eroi, iteme si alte diferențe.

Ca Aeon of Strife, Dota a permis jucătorilor să controleze o anumită campion/caracter și să intre în luptă cu echipa adversă pe cele trei benzi care unificau bazele.

Pe lângă toate acestea, Dota a introdus si modul competitiv formându-se două echipe formate din personaje controlate de jucători. De asemenea, fiecare echipă este formată din cinci jucători, ale cărei eroi se dezvoltă pe măsură ce câștigă experiență .S-a introdus și o junglă plină de creaturi ce oferă anumite bonusuri. Dota este cel mai bun concept care s-ar putea cere de la un MOBA.

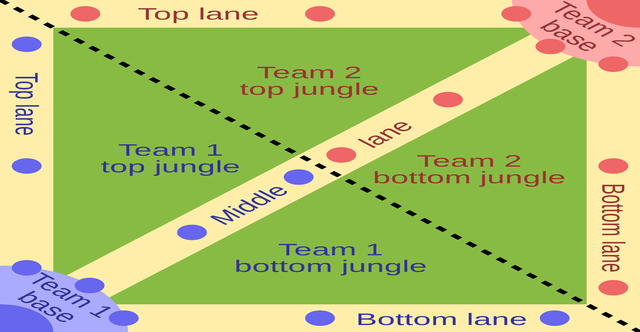
[](http://venturebeat.com/2014/09/01/the-history-of-mobas-from-mod-to-sensation/dota-allstars/)

**League of Legends (LOL)**

Acest joc video a fost dezvoltat și comercializat de către Riot Games pe platformele Microsoft Winows si Mac OS. În 2012 revista Forbes a numit League of Legends ca fiind "cel mai jucat joc pe calculator din America de Nord si Europa  din punct de vedere al orelor petrecute în acesta."  În Ianuarie 2014 peste 67 de milioane de persoane au jucat League of Legends, 27 de milioane într-o singură zi, și 7.5 milioane de jucători au fost conectați în același timp.

Jucătorii de League of Legends, au numele de "invocatori" controlând un singur caracter (numit campion), au abilități unice și alături de echipa lor trebuie să distrugă nexus-ul echipei adverse. Fiecare joc de League of Legends este unic, toți campionii încep destul de slabi, progresând în timp, acumulând aur și experiență în timpul jocului.

League of Legends a generat o comunitatea competitivă activă care este în continuă creștere. În Europa și în America de Nord, Riot Games organiează [League of Legends Championship Series](https://ro.wikipedia.org/w/index.php?title=League_of_Legends_Championship_Series&action=edit&redlink=1), care constă în întrecerea a 8 echipe profesioniste de pe fiecare continent. Competiții regionale asemănătoare există în: China, Coreea, Taiwan și Asia de Sudest. Aceste compleții regionale au apogeul la anualul [Campionat Mondial de League of Legends](https://ro.wikipedia.org/wiki/Campionatul_Mondial_de_League_of_Legends), care în 2013 a avut un premiu de 1 milion de dolari și a atras peste 32 de milioane de vizitatori online.



## Cerințe funcționale

* Crearea de cont direct din aplicație.Jocul nu se va putea juca fără autentificare
* Pentru a porni o partidă, jucătorul poate crea o cameră sau se poate alătura uneia deja create
* Conversațiile se vor desfășura pe chat global dar și pe chatul din joc.
* Posibilitatea închiderii camerei si revenirii către meniul principal
* Intrați în meci, jucătorii vor alege unitatea erou/campionul cu care vor intra în luptă din lista campionilor oferită de sistem.
* Pe parcursul jocului eroul poate fi mutat pe hartă folosind click-ul de la mouse. Zona de actiune unde a fost apăsat click-ul va deveni destinația caracterului. Unitatea se va deplasa către destinație cu o anumită viteză, acesta fiind influențată de anumite atribute
* Eroul poate urmări jucătorul advers până se ajunge in raza de actiune dupa care acesta va ataca, fie de la distantă, fie din apropiere. Pe langa atacul standard există si atacul magic. Jucătorul poate folosi anumite abilităti ce vor necesita puncte magice. Atacul va fi influențat de anumite atribute.
* Pe lângă acestea există un sistem de avansare a caracterului pe parcusul jocului si un sistem de dezvoltare a abilităților
* Fiecare actor existent pe hartă va dispune de un sistem de viața, atac
* Jucătorul poate apropia sau depărta camera atașată eroului, schimbându-se perspectiva
* Unitățile pe care fiecare jucător le deține vor ajuta la distrugerea apărării adversarului. Ele se vor deplasa pe un anumit drum prestabilit
* La finalul jocului jucatorilor li se vor memora rezultatele
* Oferirea unei statistici asupra meciurilor anterioare

## Abordare tehnică

Aplicația folosește pentru partea de client game engine-ul Unity, unde limbajul de de progrmare folosit este C#, iar pentru partea de server Node.js cu limbaj de scripting JavaScript. Comunicarea între client și server se va realiza prin biblioteca Socket.IO. Pentru a stoca date aplicatia foloseste Mysql.

**Unity**

Unity este unul dintre cele mai folosite motoare pentru jocuri 3D, dar cu acesta se pot creea fara nicio problemă si jocuri 2D. Cu ajutorul acestuia folosind același cod de bază se pot dezvolta aplicații pe mai multe platforme precum Windows, Android, iOS. Versiunea folosită în realizarea jocului este Unity 5.

**C#**

Limbajul ales pe client este C# deorece este unul dintre cele mai avansate pentru dezvoltarea de jocuri. De asemenea este un limbaj de programare simplu, modern în programare fiind orientat pe obiecte.

**Javascript**

La fel ca si limbajul C#, JavaScript poate fi folosit pentru dezvoltarea unei aplicații indiferent de platforma folosită, fiind simplu de utilizat si orientat pe obiecte. Limbajul conține o bibliotecă standard de structuri de date cum ar fi Arrays, Date, Math si un set de elemente de limbaj cum ar fi operatori, structuri de control, declarări. Javascriptul nu oferă suport rețelistic.

**Node.js**

Node.js este o platformă software ce utilizează Javascript ca limbaj de scripting și este folosită pentru a construi aplicații de rețea scalabile, în special aplicațiile de comunicare și jocurile de tip browser, deoarece amândouă necesită o comunicare în timp real între server și numărul foarte mare de clienți ce accesează aplicația.

**Socket.IO**

Socket.IO este o bibliotecă JavaScript pentru aplicații în timp real. Aceasta facilitează comunicare bidirecțională între clienți și conține două componente, biblioteca ce rulează pe partea de client și biblioteca pentru server. Exact ca și Node.js, comunicarea este bazată pe evenimente.

**MySQL**

Pentru a stoca datele aplicația folosește un sistem relațional de gestiune a bazelor de date numit MySQL, acesta fiind cel mai popular sistem open-source la ora actuală.

# Contribuție

BattleOfTitans este un joc tridimensional în care utilizatorul va putea să se împrietenească și să socializeze cu alți jucători. Totodată acestă își va putea testa abilitățile de viteză, gândire, tactică prin alăturarea la serviciul de camere, unde se va disputa meciul. Jucătorul iși va alege unitatea campion pe care va dori să o controleze în timpul partidei. Unitatea va dispune de un sistem de auto-atac dar și de un set de abilități magice.

Unitățile ajutătoare vor fi activate la un anumit interval de timp. Distrugerea acestora de către utilizatori va oferi puncte de experiență, astfel va crește nivelul campionilor. Există si un sistem de recompense, statistici și istoric al jocurilor desfășurate.

Acestă lucrare de licență este împarțită in trei componente referitoare la modul de funcționare a aplicației pe partea de client/server, concepete si algoritmi utilizați.

În prima parte sunt prezentate informații generale despre arhitectura client-server a aplicației, in a doua parte este descrisă dezvolatarea aplicației client si a jocului, iar în a treia parte dezvolatarea serverului .

Contribuții generale pentru realizarea celor mai importante facilități ale aplicației atât pe partea de client, cât și pe partea serverului:

* Implementarea sistemului de chat global și în joc
* Transpunerea hărții jocului într-un obiect bidimensional folosit la deplasarea caracterelor
* Implementarea unui algoritm de inteligență artificială pentru detectarea adversarilor
* Sistem de atac apropiere/depărtare
* Implementarea unui algoritm pentru limitarea distanței de atac a abilității principale

• Implementarea unui server bazat pe evenimente pentru a facilita jocul pe camere

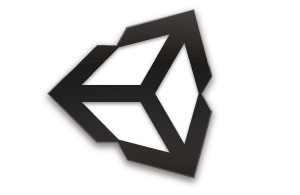
• Mecanisme pentru realizarea unui server autoritar, existența claselor pentru fiecare entitate utilizată de client

• Baza de date necesară funcționarii aplicației

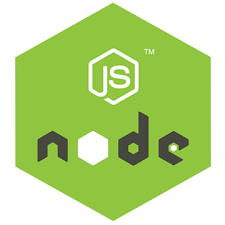
# Proiectare

## Arhitectura soluției

Aplicația este alcătuită din două proiecte, client Unity și server NodeJs, folosindu-se componenta Socket.IO pentru realizarea comunicării între cele doua părți. Pe partea de server pentru a stoca datele se foloseste o bază de date MySQL.



Unity **Client**



**Node Server**



Mysql DB



Socket.IO **Client**

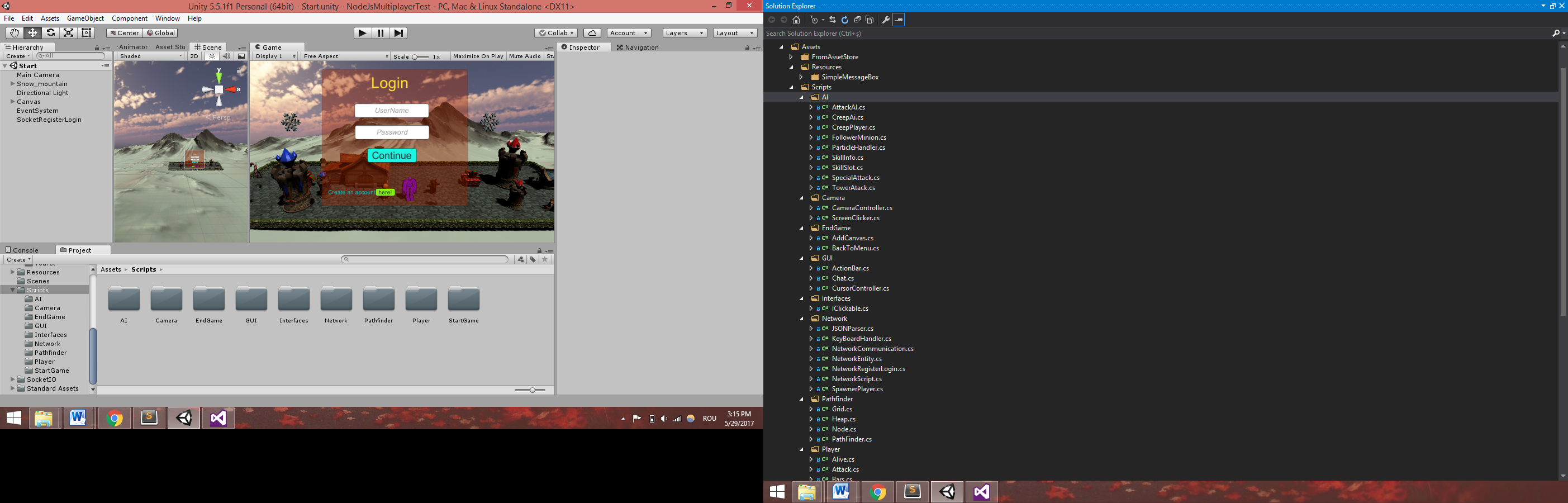
Socket.IO **Server**

s

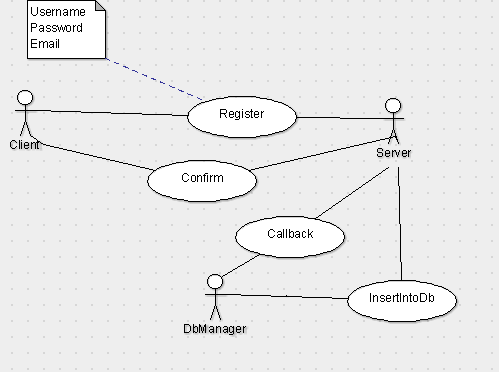
Jocul propiu-zis este dezvoltat pe partea de client si este impartit in mai multe scene :

* **Register**
* **Login**
* **Game Menu**
* **Character Selection**
* **Game**
* **End**

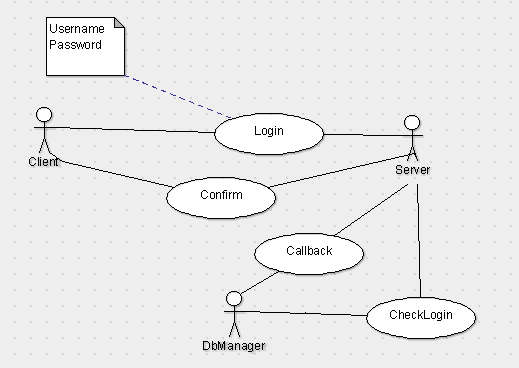
Pentru fiecare scena am realizat cate un pachet denumit sugestiv, fiecare conține clasele specifice fiecărui obiect din joc (eng:GameObject) din scena respectivă. La acestea se mai adaugă și alte pachete specifice scenei unde se desfășoară jocul . Astfel pe partea de client avem 10 pachete ce conțin un numar de ridicat de clase/scripturi (aprox 50).



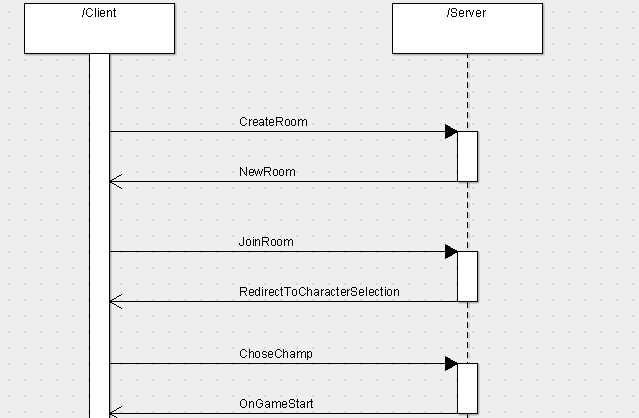
1. Register – Înregistrarea unui nou utilizator.



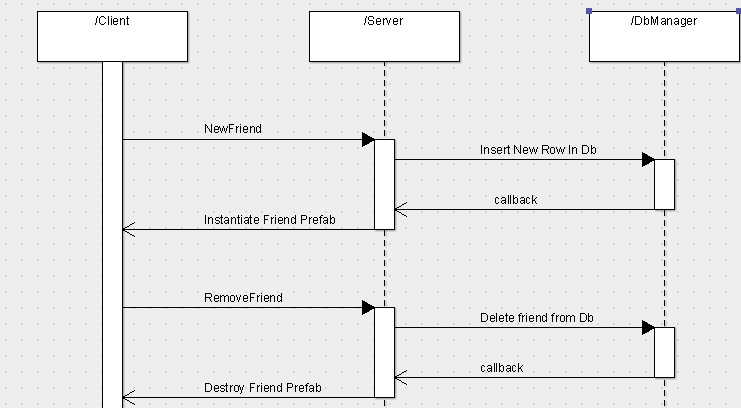
1. Login – Logare unui utilizator



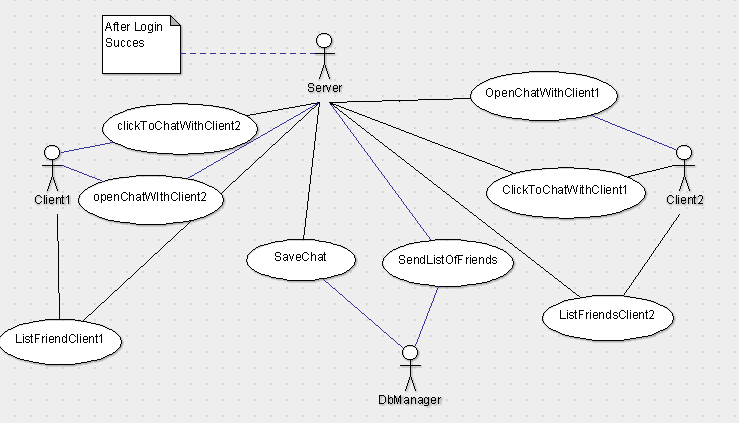
1. GameMenu – NewRoom sau JoinRoom



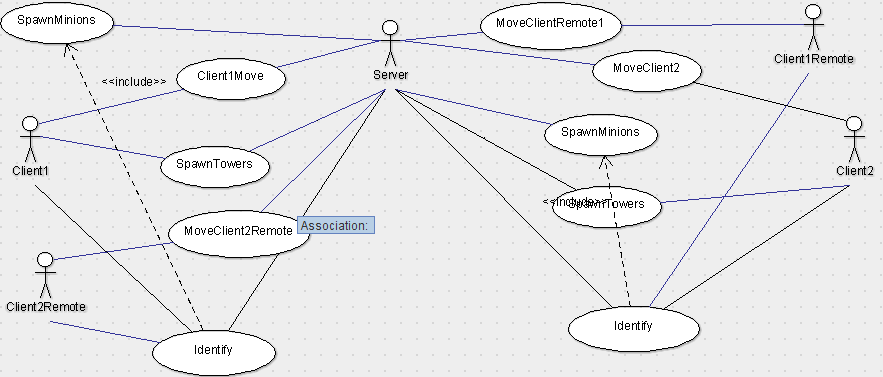
1. GameMenu – NewFriend sau RemoveFriend



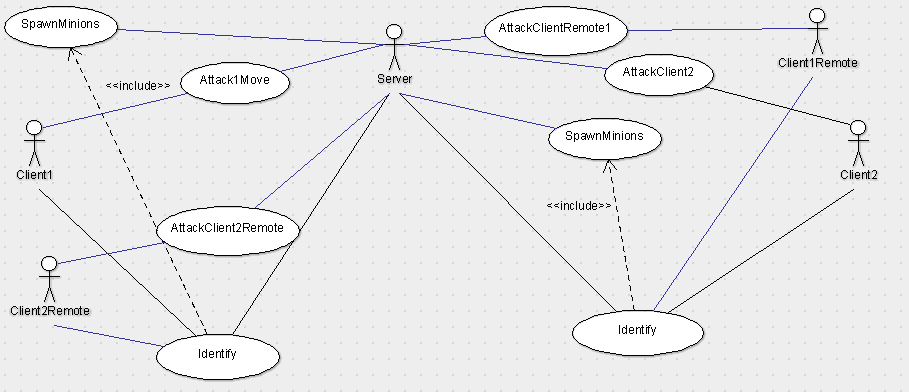
1. GameMenu – Chat între doi prieteni



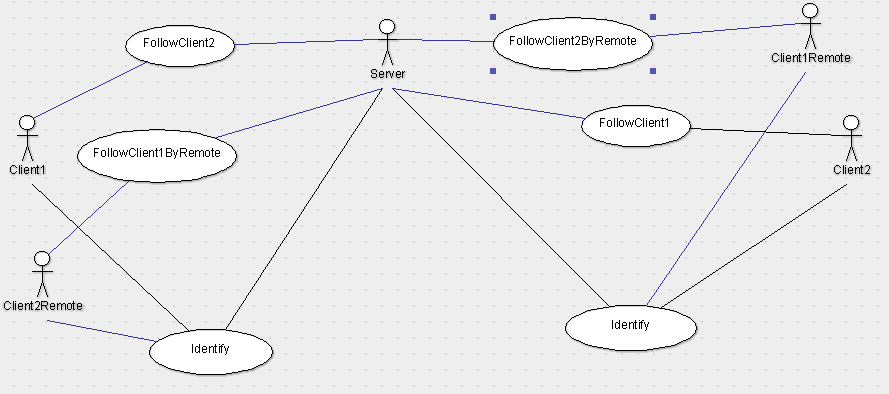
1. Game - Identificarea fiecarui jucător si sincronizarea mișcării pe hartă a acestora.Fiecare client conține si copia celuilalt fiind controlat prin intermediul evenimentelor trimise de celălalt client către server.



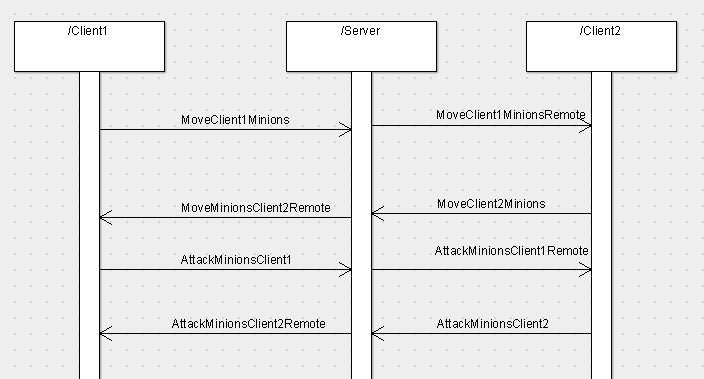
1. Game – Atac între utilizatori sincronizat prin intermediul serverului similar cu mișcarea acestora.



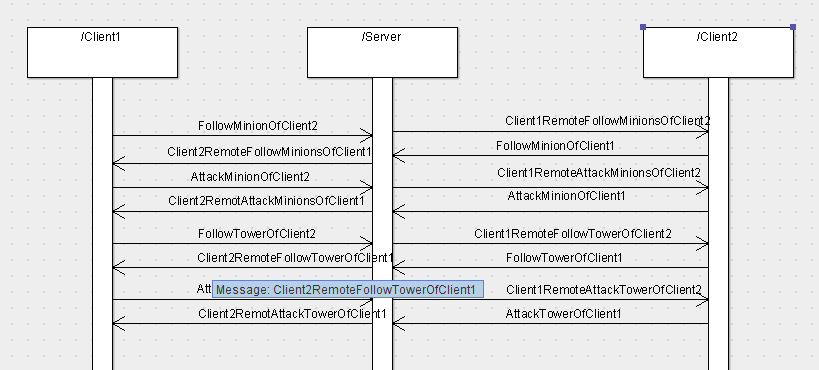
1. Game – Urmărirea jucătorilor între ei prin detectarea activității mouse-ului asupra clonei celuilalt client. Evenimentul este trimis către server sincronizându-l.



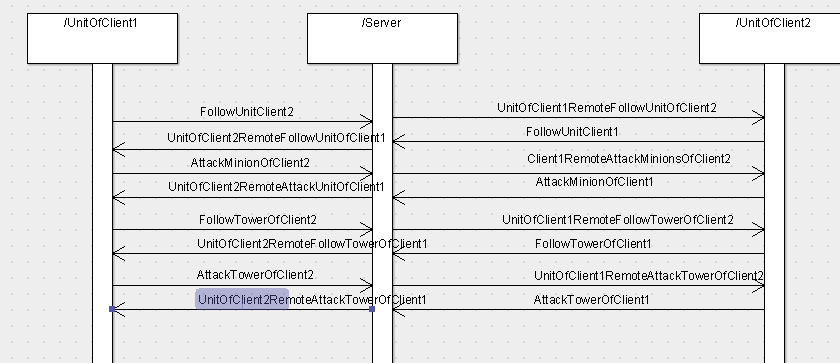
1. Game – Mișcarea unităților si atacul acestora. Am decis că fiecare client are propriile unități si clonele unităților celuilalt client. Unitățile clientului clona sunt controlate la fel ca acesta tot prin intermediul serverului.



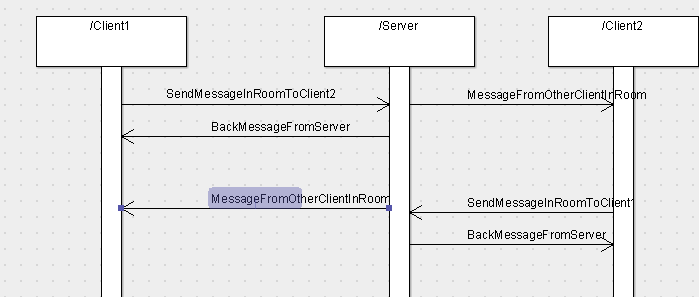
1. Game – Urmărirea unităților si atacul acestora de către clienți. Se bazează pe acelasi principiu ca la unitățile campion controlate de client.



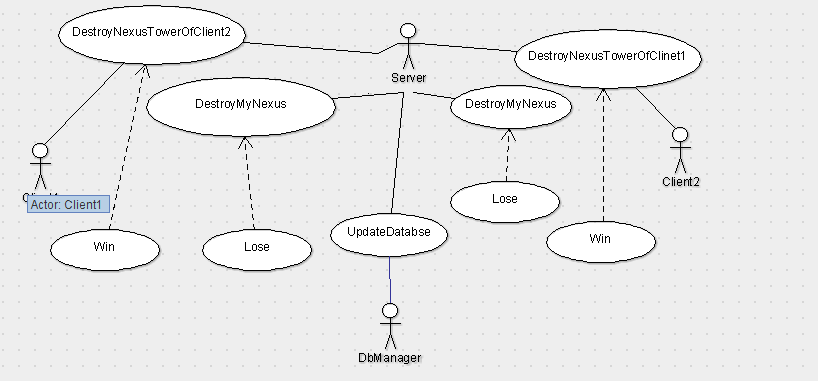
1. Game – Urmărirea unităților între ele, atacul acestora și atacul asupra turnurilor.



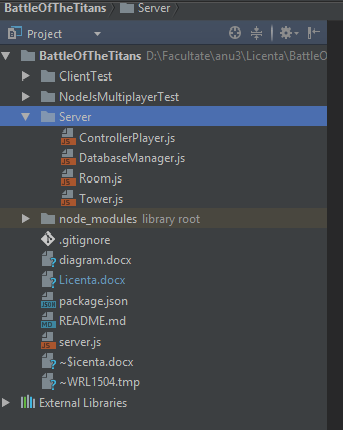
1. Game – Chat în joc



13.Game – Sfârsit joc si revenire către meniul principal



Serverul este alcătuit din 5 clase principale si va permite rularea in mod asincron a evenimentelor, comunicarea realizându-se in mod real.



**Arhitectura acestei aplicații** este astfel concepută, încat fiecare diviziune a ei să fie folosită eficient în sensul înțelegerii ușoare a datelor și a unei experiențe plăcute.

Sistemul multiplayer va fi implementat cu ajutorul Node.js-ului si a WebSocket-urilor. Ambele sunt două entități eficiente si destul de flexibile pentru crearea serverului de joc si susținerea meciurilor.

## Modelarea datelor

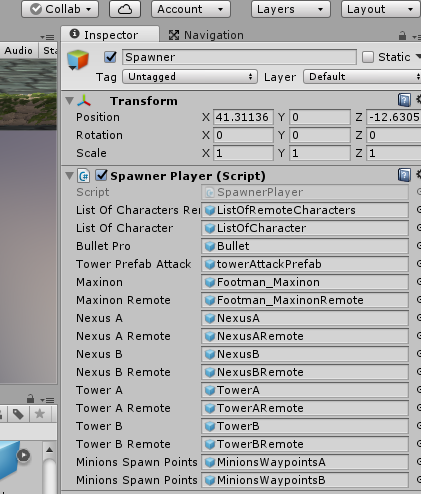
Aplicatia fiind compusă și din client și din server modelarea datelor se face in cele doua planuri. Astfel am împartit acest subcapitol in modelarea datelor pe partea de client, pentru a modela entitățile existente in joc și modelarea datelor pe partea de server, pentru a memora datele cu caracter persistent.

### Modelarea datelor pe partea de client

O noțiune foarte importantă în dezvoltarea unui joc prin intermediul game-engine-ului Unity5 este noțiune de GameObject, fiind cel mai abstractă clasă. Acesta se află în spatele oricărui obiect tridimensional aflat în scena de lucru. La acest obiect se pot adăuga anumite componente, fie cele standard oferite de motorul de dezvoltare, fie anumite scripturi/clase de tipul MonoBehaviour. Aceste scripturi trebuie implementate de dezvoltator astfel încat sa se obtină rezultatul dorit.

Pentru dezvoltarea acestui joc am modelat un număr destul de mare de entități. Unele dintre cele mai relevante entități : **Ground, Spawner, ListOfCharacters, ListOfRemotes, Network, Footman\_Maxinon,** **Footman\_MaxinonRemote**

Scripturile acestora folosesc la randul lor anumite obiecte salvate, prefabricate. De exemplu pentru enitatea Spawner:



Fiecare caracter din ListOfCharacter este compus din urmatoarele scripturi: PathFinder, NetworkCommunication, NavigateToPosition, Follower, Network Entity, Target, Alive, Attack, Special Attack x4, Mana. Ground este alcătuit din: MoveToClick, Grid, CursorController. ListOfRemotes coține tot ceea ce contine si lista principală de caracter, la acesta adăugându-se FollowClick, ce permite selectarea, în vederea urmăririi și atacării eroului advers.

Footman\_Maxion contine următoarele scripturi: Target, PathFinder, NavigateToPosition, NetworkEntity, Alive, FollowMinions, CreepAi, CreepPlayer, AttackAI, FollowerMinion. Footman\_MaxionRemote renuntă la CreepPlayer, dar i se adaugă CursorController, FollowMinions.

Aceste scripturi vor fi prezentate la secțiune de implementare.

În mare parte fiecare actor aflat in scena principală a jocului este descris de anumite propietăți : currentHp, maxHp, speedAttack, damage, speedMove, skillDamages etc.

### Modelarea datelor pe partea de server

Pentru salvarea datelor persistente ale utilizatorilor contul, parola, prietenii, statisticile meciurilor desfășurate am folosit MySQL. Baza de date este alcătuita din 2 tabele principale.

Tabela Users:

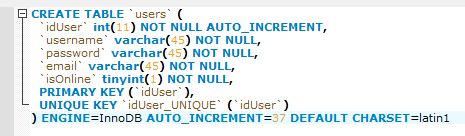
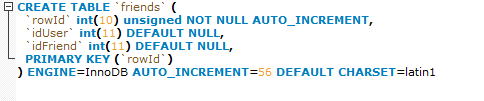


Tabela Friends:



Câmpurile tabelelor sunt denumite in mod sugestiv. Pentru un utilizator vom memora contul, parola și emailul. Vom folosi si un idUser care va permite identificarea mai ușoara atunci cand vom interoga și alte tabele.

## Protocoale de comunicare client – server

Librăria care permite comunicarea bidirecțională în timp real între clienți si server este Socket.io . Pe partea de client este instalată componenta oferită de Unity5, iar pe partea de server, cea oferită de Node.js.

Acesta ne permite să transmitem anumite evenimente, date in rețea. Pe lângă evenimentele de conectare, transmitere de mesaje, deconectare, putem emite și alte evenimente ce corespund deciziei noastre. Aplicația se bazeaza pe un numar ridicat de evenimente, specifice fiecarei scenă.

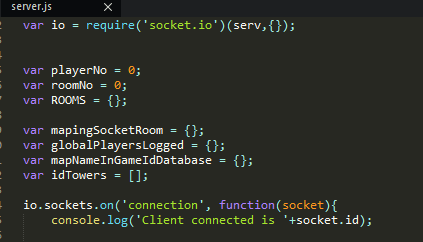
Exemplu:

Conectarea:

* la pornirea aplicației, pe partea de client



* se obține componenta socketului și se realizează conectarea cu serverul la o anumită adresă și port, stabilită din interfața programului
* la pornirea aplicației pe partea de server:

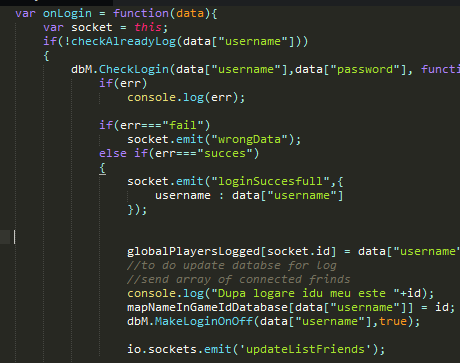


* se obține modulul oferit de node
* se asteptă conectarea clienților la un anumit port

Atunci când se dorește realizarea unui eveniment se transmit datele către server, numele evenimentului tramis de către client va trebui sa coincidă cu numele evenimentului pe partea de server. Astfel evenimentul care se emite trebuie sa aiba pereche pe partea de server. De exemplu atunci când se face logarea în sistem.



Datele trimise de la client către server sunt stocate sub forma unui dicționar. În acest scop se construiește un obiect de tip JSONObject ce este transmis in rețea. Mai multe despre JSONObject și transmiterea datelor la partea de implementare. Fiecărui eveniment recepționat, atât pe partea de client, cât și pe partea de server îi corespunde o funcție de tip callback.

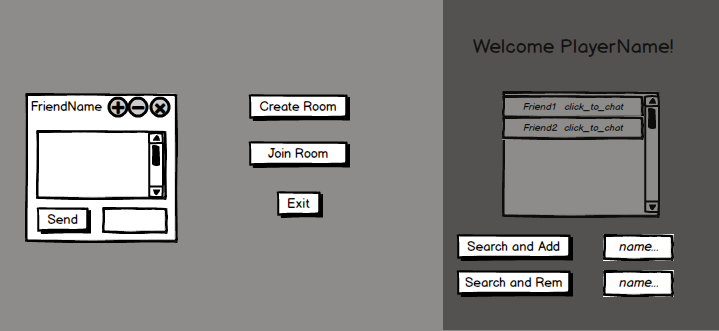
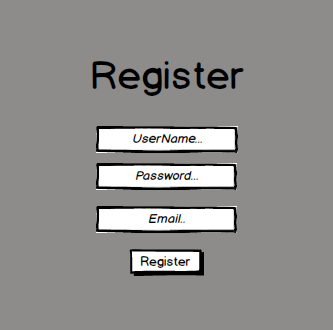
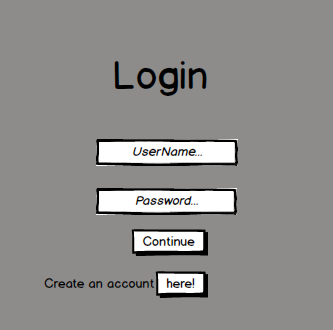


## Interfața cu utilizatorul

Proiectarea interfeței cu utilizatorul este importantă atât din punctul de vedere al interacțiunii (user experience – UX), cât și al aspectului (user interface – UI).

În general interfața unei aplicații trebuie să fie ușor de înteles și de folosit astfel încat utilizatorul să fie familiarizat cu elementele din care este alcătuită. Interfața trebuie să fie simplă.

Pentru fiecare scenă din joc am realizat câte o interfața astfel:



Prima interacțiune a utilizatorului cu aplicația se va realiza prin afișarea meniului de Login. Acest meniu conține două câmpuri nume cont și parolă, un buton de confirmare, dar și butonul care va redirecționa utilizatorul către meniul de Înregistrare, acest meniu fiind similar cu precedentul. Se obervă ca cele două meniuri sunt simpliste si intuitive .

După autentificarea cu suces utilizatorul va fi redirecționat catre meniul jocului. Acesta are la dispoziție 3 butone principale CreateRoom, JoinRoom, Exit .

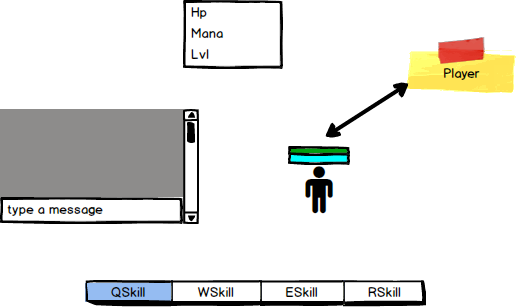
**CreateRoom** – utilizatorul va porni o cameră de joc și va astepta alăturarea unui alt client pentru a porni jocul.

**JoinRoom** – utilizatorul se va alătura unei camere deja existente, acesta primind lista de camere de la server

**Exit** – părăsirea aplicției

În partea dreaptă se află lista de prieteni ai utilizatorului. Acesta va putea porni o conversație. Prin simpla apăsare a unui prieten se va deschide o caseta de chat unde sa va putea purta discuția. Pe acestă caseta se afla 3 butone care vor acționa asupra casetei de chat: micșorare, mărire, închidere.

Când camera va fi ocupată de doi utilizatori se va porni scena de alegere a unităților campion. După confimarea celor două caractere se va porni scena jocului.



În scena jocului există o bară pentru cele patru abilități, o casetă pentru chatul din joc, o casetă pentru informațiile curente ale jucătorului. Interacțiunea cu jocul se va realiza și cu actorii din joc, obiectele tridimensionale prezente în fiecare scenă.

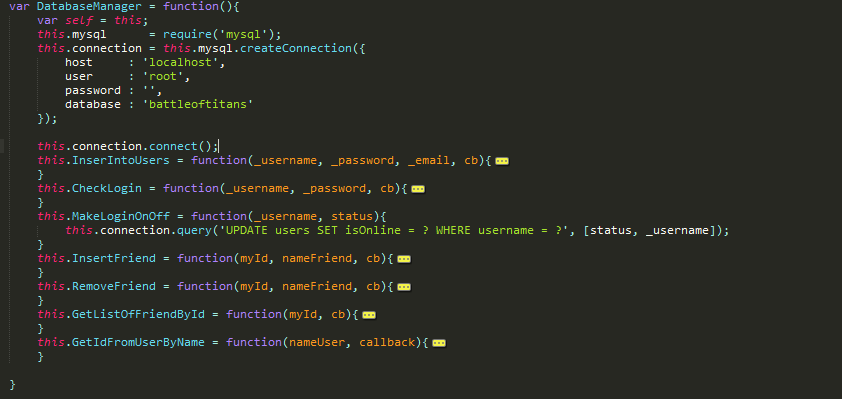
# Implementare

În acest capitol voi prezenta cele mai importante funcționalități care au condus la realizarea aplicației, dintre care amintesc:

* Realizarea chatului global și a sistemului de socializare
* Gestiunea sistemului de camere
* Algoritmul A\* Pathfinding
* Proiectarea unui alogritm de inteligență artificială pentru controlul unităților ajutătoare
* Sistemul de atac magic
* Dezvoltarea serverului semi-autoritar

## Chat

La pornirea serverului se construiește un obiect de tipul **DatabaseManager,** ce va fi responsabil cu interogările către aceasta. În momentul creării acestuia se apeleaza constructorul care va realiza conectarea către baza de date.



După logarea cu succes clientul va cere de la server lista de prieteni, pe care acesta îi deține. Acesta listă este obținută prin interogarea bazei de date. Totodată când un anumit client se loghează este trimisă către toti clienții lista pentru actualizare, în cazul in care jucatorul conectat este prieten cu aceștia.

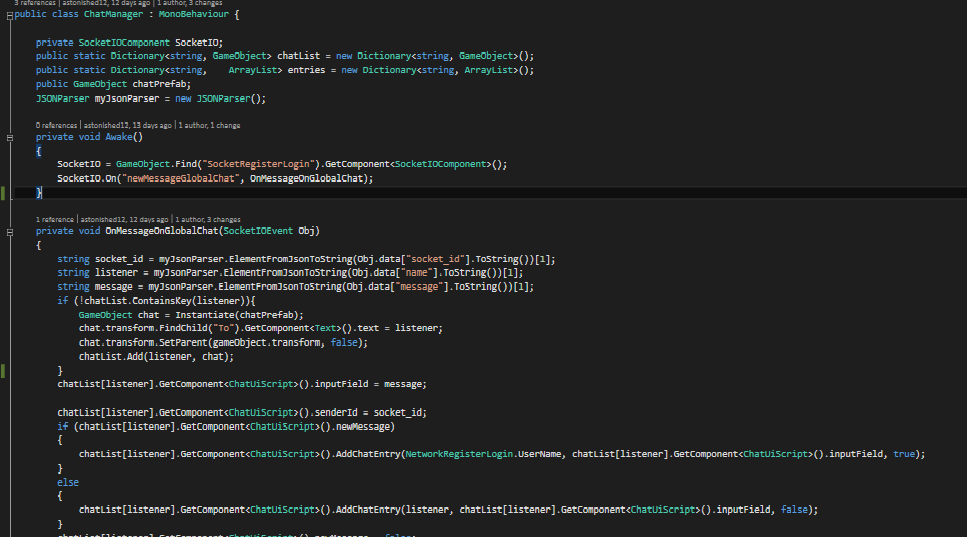
Pentru adăugarea și ștergerea unui prieten se trimite către server numele acestuia, se actualizează baza de date și se trimite evenimentul **newFriend** către cel care a realizat cererea, dar și către cel căreia i-a fost adresată. Clientul va fi notificat, la ecran se va deschide o caseta de dialog cu mesajul corespunzător.

Astfel pe fiecare client se construiește un nou obiect de tipul **friendPrefab**, obiect ce va fi adăugat in lista prietenilor. Lista prietenilor, fie va fi compusă din prieteni care sunt activi, fie din cei care nu sunt activi. Schimbul de mesaje între clienți se poate realiza numai daca aceștia sunt activi.

**ChatManager**

Pentru a reține activitatea pe care utilizatorul o va desfășură impreuna cu prietenii acestuia am construit o clasă ce va avea ca scop monitorizarea schimburilor de mesaje și a discuțiilor curente. Astfel daca o persoană vrea sa trimită un mesaj către un alt client, în momentul recepționării se va deschide o un obiect de tipul **ChatUI**, ce va fi pornit doar odată. Nu vor exista doua casete pentru aceeași discuție. Acest lucru este valabil și pentru schimbul de mesaje în sens invers.

Se vor memora intrările noi ale mesajelor iar pentru obiectul de tip chat se va actualiza lista de mesaje.

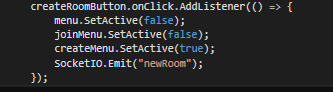
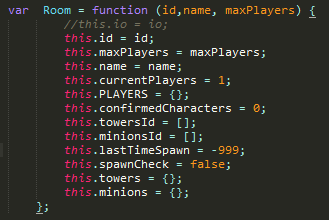


Conversațiile nu vor fi salvate deoarece chatul a fost proiectat în scop informativ în legătura cu jocul deoarece am dorit să nu se realizeze un mediu de socializare.

## Sistemul de camere

Pentru a disputa o partidă clientul fie isi creează o cameră nouă în care va astepta ca un alt utilizator să i se alăture, fie se va alătura unei camere deja existente.

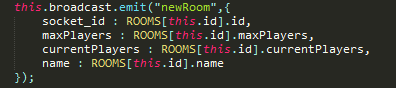
În cazul în care acesta dorește să creeze o cameră nouă, va acționa butonul NewRoom din meniul principal al jocului. Odată apăsat butonul se va apela o funcție care va trimite către server faptul că utilizatorul curent dorește să deschidă o nouă camera. Serverul va prelua acest semnal și va memora camera.



Se va seta deținătorul camerei, se va memora numele camerei, se vor genera identificatorii turnurilor folosindu-se de modulul **shortid** oferit de node.js. Camera se va memora in dicționarul global **ROOMS**, memorare ce va avea cheia id socketului utilizatorului. Clientul va fi identificat cu socketul înca de la conectarea la server. Folosind acest lucru am alăturat socketul cu camera proaspăt creată. Se va construi si un obiect de tip **ControllerPlayer**, ce va fi adăugat in lista jucătorilor camerei. Acest lucru va ajuta la implementarea autorității serverului.



Serverul va trimite către toti utilizatorii conectați faptul ca s-a deschis o nouă cameră, trimițandu-le datele despre acesta.



Clientii vor prealua la rândul lor evenimentul transmis de către server și vor actualiza dicționarul global al camerelor. În cazul in care aceștia acționeaza butonul de **JoinRoom,** se va transforma dicționarul camerelor în obiecte de tip UI, ce vor fi aduse în interiorul canvasului meniului principal.



Daca se dorește alăturarea la o anumită camera se va transmite către server evenimentul de **joinRoom,** se va actualiza camera respectivă, mărindu-se numarul de jucători, socketul clientului se va alătura si el camerei, după care este trimis catre cei doi participanți evenimentul **joinSuccesFull**.



În momentul recepționării acestui eveniment de către jucătorii din camera respectivă, aceștia vor fi redirecționati către scena alegerii campionilor cu care vor intra în luptă.

## Algoritmul A\* Pathfinding

### Introducere

Am folosit acest algoritm pentru a realiza mișcarea caracterului în timpul jocului, deoarece acesta este unul dintre cei mai populari algoritmi pentru cautarea traseelor.

Algoritmul oferă cel mai scrut drum dintre doua puncte aflate pe hartă. Jocul propune o hartă în manieră 3D, dar se va utiliza doar sistemul cartezian. Se ignoră astfel coordonata Y a sistemului. Harta va conține anumite obstacole ce vor fi luate în considerare atunci când se va stabili drumul minim.

**Câteva propietăti ale algorimtului A\***

* este complet mereu va găsi o solutie dacă acesta există
* poate folosi o strategie care să marească viteza de executare a procesului de cautare
* nodurile pot avea diferite costuri , există posibilitatea ca unele noduri sau drumuri să fie mai dificile de traversat.
* dacă se dorește acesta poate căuta în mai multe direcții

Pentru a putea aplica algoritmul de cautare am transformat hărta intr-un obiect matricial, unde fiecare element este de tipul **Node**.

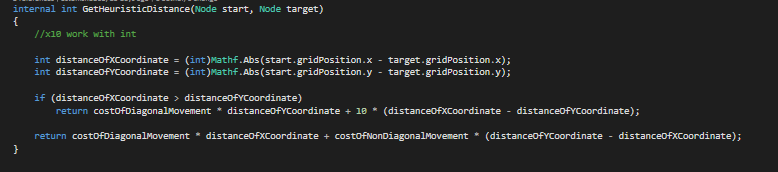


Pentru fiecare nod in parte trebuie stabilitate urmatoarele propietăți:

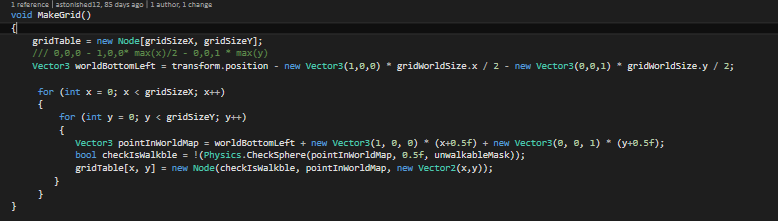
* **parintele**: referință ce oferă posibilitatea întoarcerii înapoi către nodul de start
* **costul G**: este costul de bază al nodului și este calculat ca fiind costul total de la nodul de start către nodul curent
* **costul H**: este costul oferit de euristica alesă. Acesta este cel mai important cost deorece influentează modul de deplasare.
* **costul F**: este costul total al drumul prin nodul curent și are drept valoare suma dintre costul G și costul H.

Punerea în aplicare a scorului H poate varia în funcție de proprietățile grafului căutat. Cele mai utilizate heuristici sunt: Distanța Manhattan, Distanța Euclidiană, Distanța Diagonală având cost uniform, Distanța Diagonală.

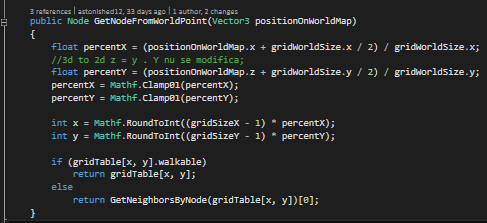
Algoritmul implementat în cadrul jocului BattleOfTitans folosește ca euristică Distanța Diagonală, ce ce va avea costul pe diagonală 14, iar pentru restul direcțiilor 10.



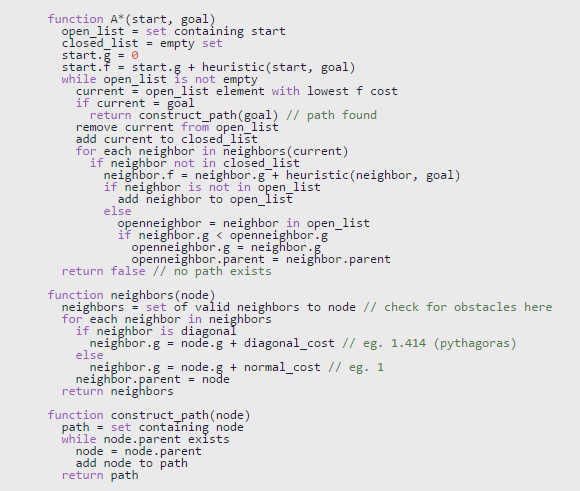
Astfel harta va avea ca și componentă principală scriptul **Grid**  unde se va realiza împartirea în noduri. Se calculează punctul aflat in stânga sus al hărții, după care are loc divizarea în cele **gridSizeX\*gridSizeY** componente obiect. Pentru fiecare astfel de componentă se verifică posibilitatea de acces asupra acestia, se construieste un obiect de tipul **Node** și se adaugă la obiectul matricial.



Având acest obiect matricial putem verifica de fiecare dată ce se află pe hartă la anumite coordonate. Pentru a stabili destinația unde trebuie sa ajungă caracterul calculam indecsi matricii pornind de la coordonatele nodului referință. Astfel se face o aproximare procentuală a celor doua coordonate. În cazul in care nodul care trebuie returnat are propiretatea de accesare, acesta se va utiliza ca destinație finală pentru alogritmul A\*. În cazul contrar este returnat primul vecin al acestuia, fiind cea mai apropiată distanțâ. Acest lucru este mentinut de structura Heap pe care se bazează fiecare cautare.



**Pseudocod**



### Binary Heap

Un **binary heap** este un arbore binar cu următoarele proprietăți.

* Este un arbore complet având toate nivelele complete, cu excepția posibilă a ultimului nivel. Acestă structură oferă posibilitatea stocării elementelor intr-un vector.
* Strcutura poate fi de tipul **MinHeap**  sau **MaxHeap**. Într-o structura de tipul MinHeap elementul rădăcină este minumul dintre toate cheile prezente în vector. Aceeași proprietate trebuie să fie adevărată recursiv pentru toate nodurile din structură. MaxHeap este similar cu MinHeap.

În cadrul jocului BattleOfTitans avem nevoie de o structură de tipul MinHeap în care vom reține nodurile cele mai apropiate de fațsă de punctul de căutare.

Pentru a accesa indexul copilului stâng al nodului ce are indexul părinte **p** se poate folosi formulei 2 \* **p**, în timp ce indexul copilului drept cu formula 2 \* **p** + 1. De asemenea, indexul unui părinte poate fi găsit cu [**p** / 2]. Nodul rădăcină are indexul 1. Următoare poziție liberă este dată de **numarul de elemente** + 1.

Există două operațiuni de bază care pot fi executate aspura unui structuri Heap:

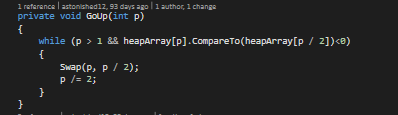
* Introducerea unui nou element in Heap
* Găsirea și eliminarea elementului minim din structură

Conceptual, ambele operații sunt destul de simple. Dar, la fel ca în cazul arborilor AVL, oricare dintre acestea poate determina încălcarea proprietăților de heap.

**Inserarea**

Pasul 1: Se poziționează nodul în prima poziție liberă din structură

Pasul 2: Se compară nodul nou introdus cu părintele său. Dacă nodul nou introdus este mai mic, se interschimbă cu acesta.

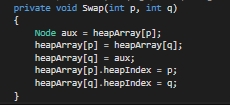


Pasul 3: Se continuă pasul 2 până când se restabilește proprietatea heap-ului.

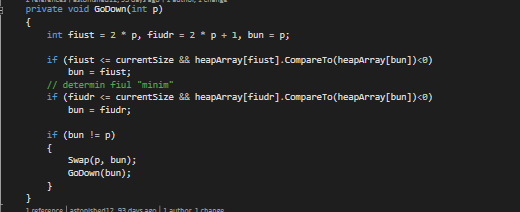
**Extragerea minimului**

Pasul 1: Se obține nodul minim.

Pasul 2: Se înlocuiește nodul minim cu ultimul nod aflat pe ultimul nivel, astfel se stabilește un nou nod rădăcină.

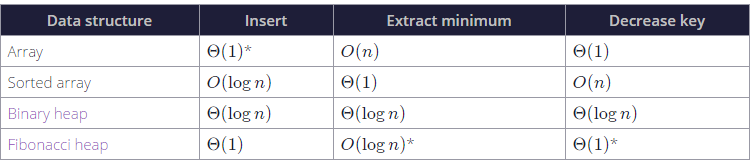


Pasul 3: Se compară nodul rădăcină cu copiii acestuia. Dacă unul dintre copii este mai mic, se înlocuiește cu cel mai mic copil.



Pasul 4: Se repeta pasul 3 până când se restabilește proprietatea heap-ului.

**Tabelă complexități**



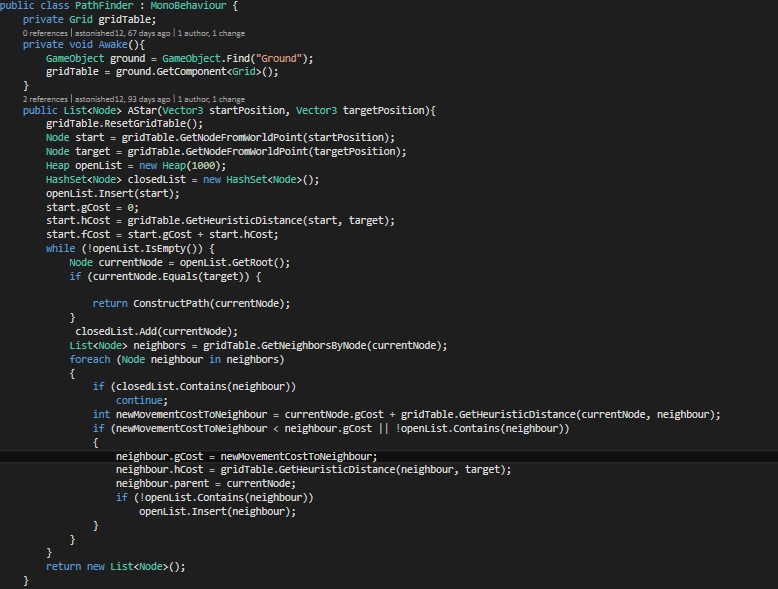
### Aplicare algoritm

Algoritmul presupune crearea a două liste:

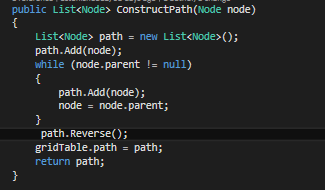
* listă cu nodurile nevizitate (en. Open List)
* listă cu nodurile vizitate (en. Closed List).

Prima listă se va utiliza pentru a reține cele mai bune noduri candidat pentru traversare, noduri care nu au fost luate în considerare până în acel moment.Se va începe cu nodul de start. Dacă lista rămâne goală, atunci nu există o cale de traversare posibilă.

Cea de-a doua listă începe goală și va conține toate nodurile care au fost vizitate. Principala buclă a algoritmului selectează nodul cu cel mai mic cost de a ajunge la destinație din lista cu nodurile nevizitate. Dacă acesta nu este nodul destinație, atunci algoritmul îi adaugă toți vecinii valizi în lista cu noduri nevizitate și repetă procesul.

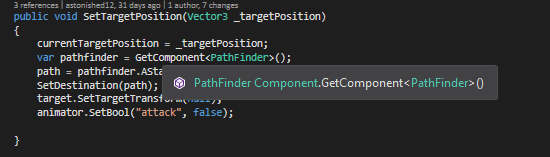


Clasa PathFinder va fi folosită pentru a determina care este drumul minim între poziția unde se află caracterul principal și destinația acestuia. De fiecare dată când alogritmul pornește se vor reseta scorurile calculate pentru fiecare nod. Pentru lista cu nodurile nevizitate se menține structura de Heap.

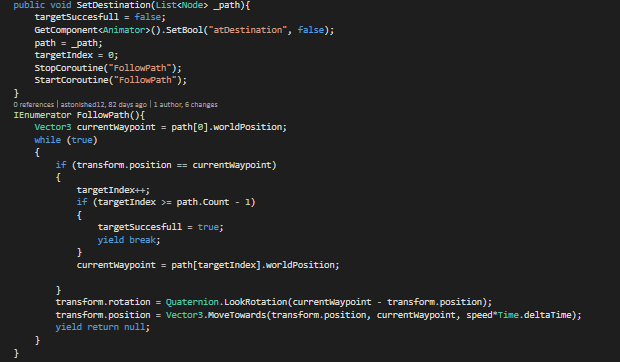


Când nodul curent va corespunde cu nodul destinație finală atunci va fi apelată metoda **ConstructPath**.

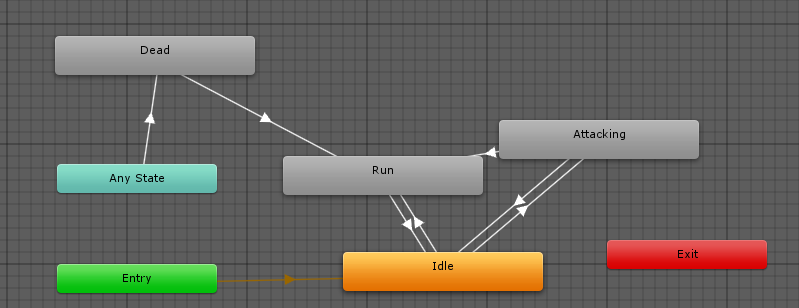
Pentru a deplasa unitatea erou am realizat clasa **NavigateToPosition** ce se va folosi de rezultatul algorimtul A\*. Se va seta drumul și se va începere parcurgerea acestuia.



În cadrul functie SetDestination se va declanșa o functie de tipul **coroutine** ce va permite orientarea și deplasarea unității către ultimul nod din lista drumului. O corutină este ca o funcție ce are capacitatea de a întrerupe execuția programului principal și de a oferi control înapoi către Unity, dar va putea de asemenea ca la urmatorul cadru să continue execuția de unde a ramas.



**Animatorul unității campion**

****

Pentru a realiza animația caracterului s-a folosit componenta **Animator**. Acest mecanism de control se bazeaza pe un automat, unde fiecare stare este reprezentată de o componentă de tipul **Animation.** Animatorul va permite declanșarea unei animații în baza unui anumit eveniment. Pentru a porni animația de mișcare se va seta flagul **atDestination** fals.

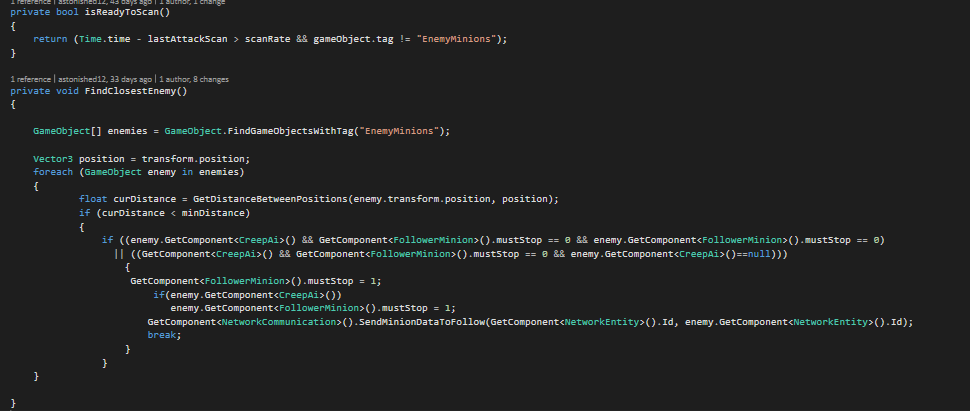
Atacul unității se poate realiza doar dacă adversarii se află la o anumita distanța minimă. Atunci când se interacționează cu aceștia există și posibilitatea de urmărire. Astfel unitatea erou se va apropia de adversar pentru a intra în raza de atac. O acțiune poate fi intreruptă de o alta. Animatorul este construit pentru a facilita o desfășurare cât mai cursivă a evenimentelor. Se poate obersva legatura dintre cei patru pivoți ai jocului: **Idle**, **Run**, **Attack** și **Dead.**

## Actori inteligenți

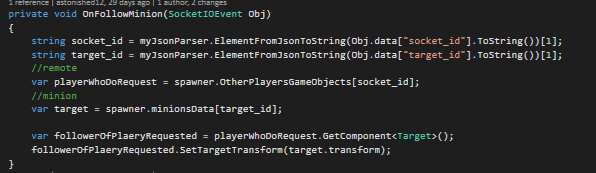
În joc pe lângă unitatea campion există unitățile ajutătoare care vor fi create la un anumit interval de timp dar și turnurile de apărare. Unitățile și turnurile sunt indentificate în rețea pe baza scriptului **NetworkEntity**.

Atunci când serverul va trimite evenimentul **spawnMinions** în camera unde se desfășoră meciul, fiecare client va adăuga trei minioni activi si trei minioni pasivi. Primii trei vor fi cei care vor detecta adversarii și rezultatele acestor vor fi folosite pentru a comunica cu serverul. Ceilalți trei vor fi minioni adversari ce vor fi controlați la distanță prin intermediul serverului. Scopul unităților este de a ajuta la distrugerea strcuturii nexus a adversarului.

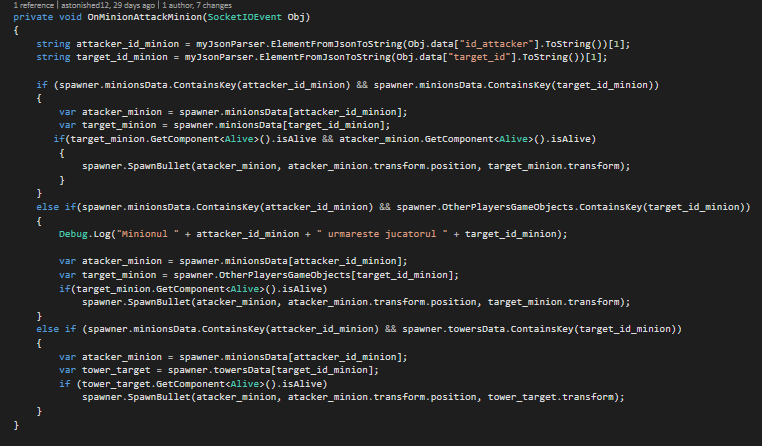
Minioni activi sunt caracterizați de scripturile **CreepPlayer** și **AttackAI**. Acestea sunt folosite pentru a detecta și ataca adversarii în timpul deplasării acestora. Atunci când in raza de acțiune a fiecărui minion apar enități inamice se va trimite către server id-ul acestuia dar și id-ul posibilei ținte. Serverul va trimite înapoi către clienți faptul că a luat la cunoștință.Pe partea de client se vor memora enitățile care au intrat in conflict și se va desfășura lupta între acestea. Atacul trece și el de la client la server pentru a memora activitatea și a susține autoritatea.



Adversarii posibili ai minionilor sunt celelalte obiecte-actor ale adversarului: unitatea erou, turnurile acestuia, minionii adversari.



Odată stabilită ținta va intra in rol scripturi AttackAI ce se va ocupa cu interacțiunea dintre atacator și tintă. În cazul în care posibila țintă a ieșit din raza de acțiune, minionul își va continua drumul către destinația lui finală. Totodată la un anumit interval de timp se resetează flagurile de urmărie și atac.



Turnurile de control se comporta asemănator cu unitățile cu excepția faptului ca ele nu se deplsează. Există trei turnuri active și trei pasive . Primele trei nu trebuie sa scaneze dupa posibili adversari in funcție de aliasul **EnemyMinions** la felca minionii pentru a crea vectorul de adversari.

Pentu a detecta posibilii adversari se folosește componenta **BoxCollider** oferită de Unity. Astfel daca actorii adversari intra în zona turnului ei vor fi detectați prin suprapunerea unei sfere cu acesția. Detectarea acestora se va realiza la 360 de grade si pe o anumita distanță oferită de raza sferei. În cazul în care există coliziuni cu sfera se va parcurge vectorul de tipul **Collider** pentru a obține obiectul aflat la distanța minimă față de turn. Dupa obținearea acestuia se va trimite către server id turnului și id-ul obiectul urmărit.



## Atac magic

Fiecare jucător deține un numar de patru abilități ce vor fi folosite pentru a ataca adversarul. Aceste abiliăți sunt poziționate .

## Autoritatea serverului