UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

Facultatea de Informatică

Lucrare de licenţă

BattleOfTitans

Propusă de

Cehan Dan Ştefan

Sesiunea: Iulie, 2017

Coordonator Ştiinţific:

Asistent, dr. Vasile Alaiba

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

Facultatea de Informatică

BattleOfTitans

Cehan Dan Ștefan

Sesiunea: Iulie, 2017

Coordonator Ştiinţific:

Asistent, dr. Vasile Alaiba

DECLARAŢIE PRIVIND ORIGINALITATEA 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

1.1 Motivație

Această lucrare de licență are ca scop oferirea unui serviciu online de amuzament sub forma unui joc video de strategie în manieră 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 unor strategii în timpul jocului și adaptarea lor la fiecare pas. Ea va fi disponibilă doar pe *Microsoft Windows*.

1.2 Context

Multiplayer online battle arena este un subgen al jocurilor video de strategie în timp real, unde jucătorul controlează un singur personaj, plasat într-una dintre cele două echipe existente. Obiectivul este distrugerea structurilor inamice cu ajutorul unităților generate periodic, unitați ce au un scop și 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, abilități noi, bani, echipament și vor putea distruge anumiți monștri din junglă. *MOBA* este o fuziune între un joc de acțiune și un joc de strategie în timp real, unde jucătorii nu pot construi clădiri sau unități.

Din dorința de a implementa un joc din această categorie am hotârat să dezvolt aplicația *BattleOfTitans*. Acesta este un joc în manieră 3D, fiind inspirat din jocurile *LeagueOfLegends*[1] *și Dota2*[2], unde utilizatorul se află în competiție cu alți protagoniști. Competiția se va realiza in manieră *1vs1*, fiecare utilizator având control asupra unei unități erou ce va dispune de anumite abilități magice. Fiecare jucător va fi poziționat intr-o bază, lângă construcția principală. De asemenea, fiecare jucător este ajutat de anumite unități ce vor conlucra la distrugerea turnurilor oponentului. Cel care distruge primul construcția finală a adversarului va câștiga.

[1] Pagina oficială http://na.leagueoflegends.com/
[2] Pagina oficială http://www.dota2.com/international/battlepass/

Aeon of Strife

În 1998, un moderator numit Aeon644 a creat *Aeon of Strife*, o hartă personalizată pentru jocul în timp real *Starcraft* al celor de la *Blizzard*. Pe acestă hartă, jucătorii controlau un singur personaj și se luptau cu o echipă adversă, ale carei unități erau controlate de calculator și 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 observa anumite diferențe notabile față de un adevărat joc *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, o echipă de eroi controlați de jucători se confrunta cu o echipă de personaje controlate de calculator. In plus, eroii nu se dezvoltau în timp ce jocul progresa, nu exista jungla suplimentară și drumurile auxiliare între cele trei benzi.

Defense of the Ancients (DOTA)

În anul 2002 cei de la *Blizzard* au lansat pe piata următorul lor joc, *Warcraft III*. La fel ca și Starcraft, acesta a venit cu anumite unelte care permiteau utilizatorilor să creeze hărți 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)*. În perioada următoare, ceilalți jucători au creat versiunile lor, fiecare adăugând proprii lor eroi, iteme și alte diferențe.

La fel ca *Aeon of Strife*, *Dota* a permis jucătorilor să controleze o anumită unitate campion/personaj și să intre în luptă cu echipa adversă pe cele trei benzi care unificau bazele.

Pe lângă toate acestea, *Dota* a introdus și modul de joc competitiv formându-se două echipe alcătuite din personaje controlate de jucători. De asemenea, fiecare echipă este formată din cinci jucători, ai cărei eroi se dezvoltă pe măsură ce câștigă puncte de experiență. S-a introdus și o junglă plină de animale magice ce oferă anumite bonusuri. *Dota* este cel mai bun concept *MOBA*.

Figura 1: Imagine din Dota

League of Legends (LOL)

Acest joc video a fost dezvoltat și comercializat de către *Riot Games* pe platformele *Microsoft Windows* și *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 personaj (numit campion), au abilități unice și, alături de echipa lor, trebuie să distrugă nexus-ul echipei adverse. Fiecare meci de League of Legends este unic, toți campionii încep de la un nivel slab, progresând în timp, acumulând aur și experiență în timpul jocului.

League of Legends a generat o comunitate competitivă activă care este în continuă creștere. În Europa și în America de Nord, Riot Games organiează League of Legends Championship Series, care constă în întrecerea a 8 echipe profesioniste de pe fiecare continent. Competiții regionale asemănătoare există în: China, Coreea, Taiwan și Asia de Sud-Est. Apogeul acestor competiții regionale este reprezentat

de Campionatul Mondial de League of Legends, organizat anual. În 2013, a avut un premiu de 1 milion de dolari și a atras peste 32 de milioane de vizitatori online. (1)

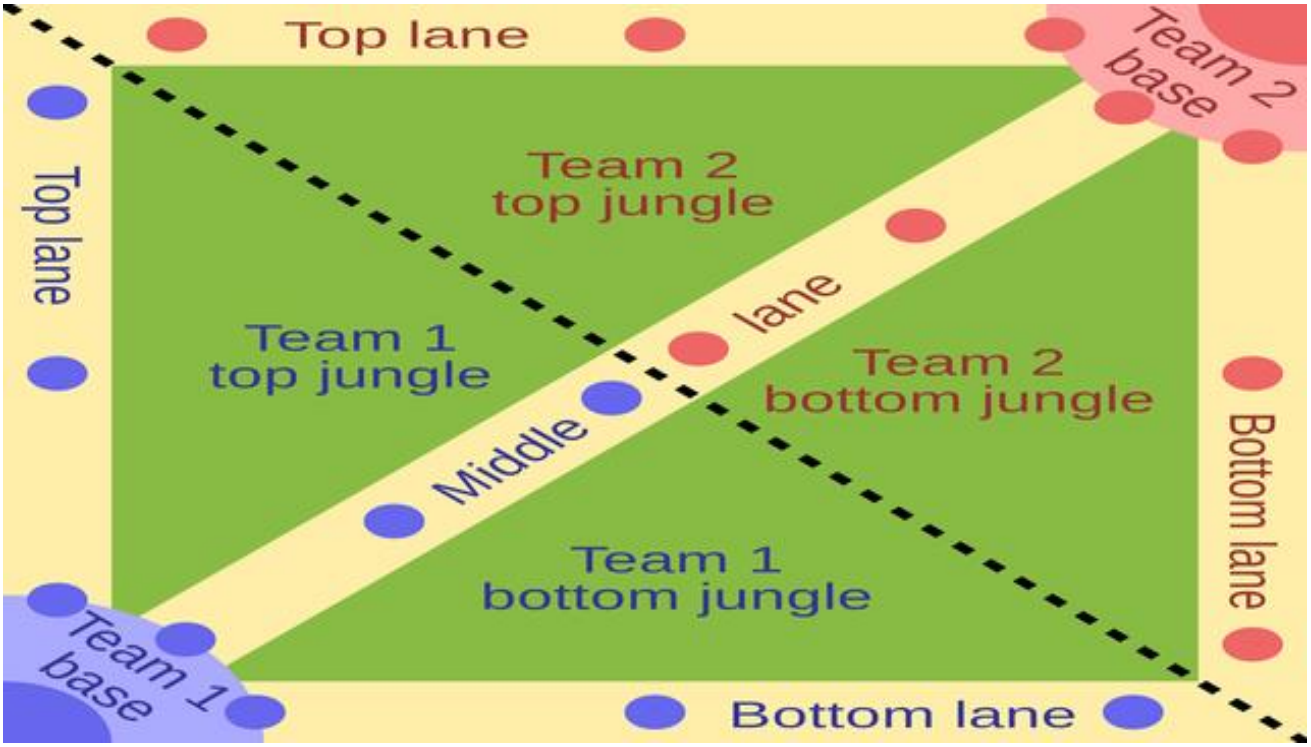


Figura 2: Hartă joc 5v5

1.3 Cerințe funcționale

- Crearea de cont direct din aplicație. Aplicația nu se va putea folosi fără autentificare
- Pentru a porni o partidă, jucătorul poate crea o cameră sau se poate alătura uneia deja create
- Conversațiile se vor desfășura în chat global dar și în chatul din joc
- Posibilitatea închiderii camerei și revenirii către meniul principal
- Înainte de 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 acțiune unde a fost apăsat click-ul va deveni destinația personajului. Unitatea se va deplasa către destinație cu o anumită viteză, acesta fiind influențată de anumite atribute

- Eroul poate urmări jucătorul advers până ajunge în raza de acțiune, după care acesta va ataca, fie de la distantă, fie din apropiere. Pe lângă atacul standard, există și atacul magic. Jucătorul poate folosi anumite abilități ce vor necesita puncte magice. Atacul va fi influențat de anumite atribute la fel ca și mișcarea
- Fiecare actor existent pe hartă va dispune de un sistem de viața, atac
- Jucătorul poate apropia sau depărta camera atașată eroului, in acest mod, schimbându-se perspectiva
- Unitățile pe care fiecare jucător le deține vor ajuta la distrugerea rezistenței adversarului. Ele se vor deplasa pe un anumit drum prestabilit
- La finalul jocului, jucătorilor li se vor memora rezultatele
- Oferirea unei statistici asupra meciurilor anterioare

1.4 Abordare tehnică

Aplicația folosește pentru partea de client game, engine-ul *Unity*, unde limbajul de de programare 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[3]

Unity este unul dintre cele mai folosite motoare pentru jocuri 3D. De asemenea, cu acesta se pot crea fară nicio problemă și jocuri 2D. Cu ajutorul acestui motor, folosind același cod de bază, se pot dezvolta aplicații pe mai multe platforme precum *Windows, Android, iOS*. Versiunea folosită în realizarea jocului este *Unity5*.

[3] Pagina officiala https://unity3d.com/

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, fiind orientat pe obiecte.

Javascript

La fel ca și limbajul *C#*, *JavaScript* poate fi folosit pentru dezvoltarea unei aplicații indiferent de platforma folosită, fiind simplu de utilizat și orientat pe obiecte. Limbajul conține o bibliotecă standard de structuri de date cum ar fi *Arrays, Date, Math* și un set de elemente de limbaj, cum ar fi operatori, structuri de control, declarări. Javascript 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, dar și jocurile de tip browser, deoarece amândouă necesită o comunicare în timp real între server și numărul mare de clienți ce accesează aplicația. (2)

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 la ora actuală.

2 Contribuție

BattleOfTitans este un joc tridimensional, în care utilizatorul va putea să se împrietenească și să socializeze cu alți jucători. Totodată, acestă va testa abilitățile de viteză, gândire, tactică ale jucătorului în timpul meciurilor, prin alăturarea la serviciul de camere. Clientul va alege unitatea campion pe care va dori să o manevreze în timpul meciului. 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. Utilizatorii pot observa statisticile și istoricul jocurilor desfășurate.

Acestă lucrare de licență este împarțită în trei componente referitoare la modul de funcționare a aplicației pe partea de client/server, la conceptele și algoritmii utilizați.

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

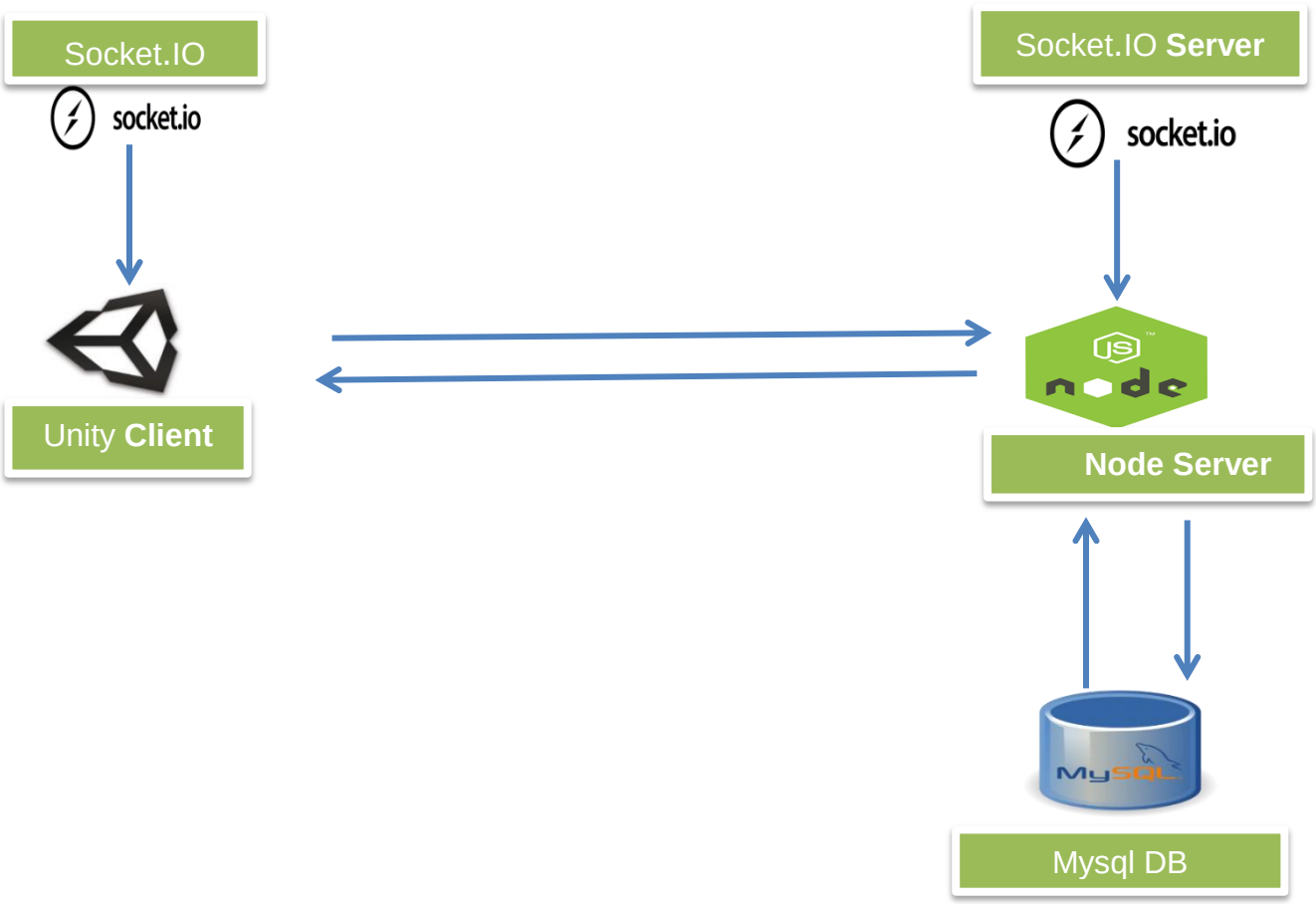
Contribuțiile generale, pentru realizarea celor mai importante facilități ale aplicației, atât pe partea de client, cât și pe partea serverului, sunt:

- Implementarea sistemului de chat global și în joc
- Transpunerea hărții jocului într-un obiect bidimensional folosit la deplasarea personajelor
- Implementarea unui algoritm de inteligență artificială pentru detectarea adversarilor
- Sistemul de atac apropiere/depărtare al campionului
- Implementarea unui algoritm pentru limitarea distanței de atac a abilității principale
- Implementarea unui server bazat pe evenimente, împărțirea serverului în camere
- Mecanisme pentru realizarea unui server autoritar, existența claselor pentru fiecare entitate utilizată de client
- Baza de date necesară funcționarii aplicației

3 Proiectare

3.1 Arhitectura soluției

Aplicația este alcătuită din două proiecte, client *Unity* și server *NodeJs*, cele doua folosesc componenta *Socket.IO* pentru comunicărea între ele. Pe partea de server pentru a stoca datele se folosește o bază de date MySQL.



F Figura 3: Arhitectura aplicației

Jocul propriu-zis este dezvoltat pe partea de client și este împărțit în mai multe scene :

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

Pentru fiecare scenă, am construit câte un pachet denumit sugestiv, fiecare conținând clasele specifice obiectelor din joc (en: *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 număr destul de ridicat de clase/scripturi (aprox 50).

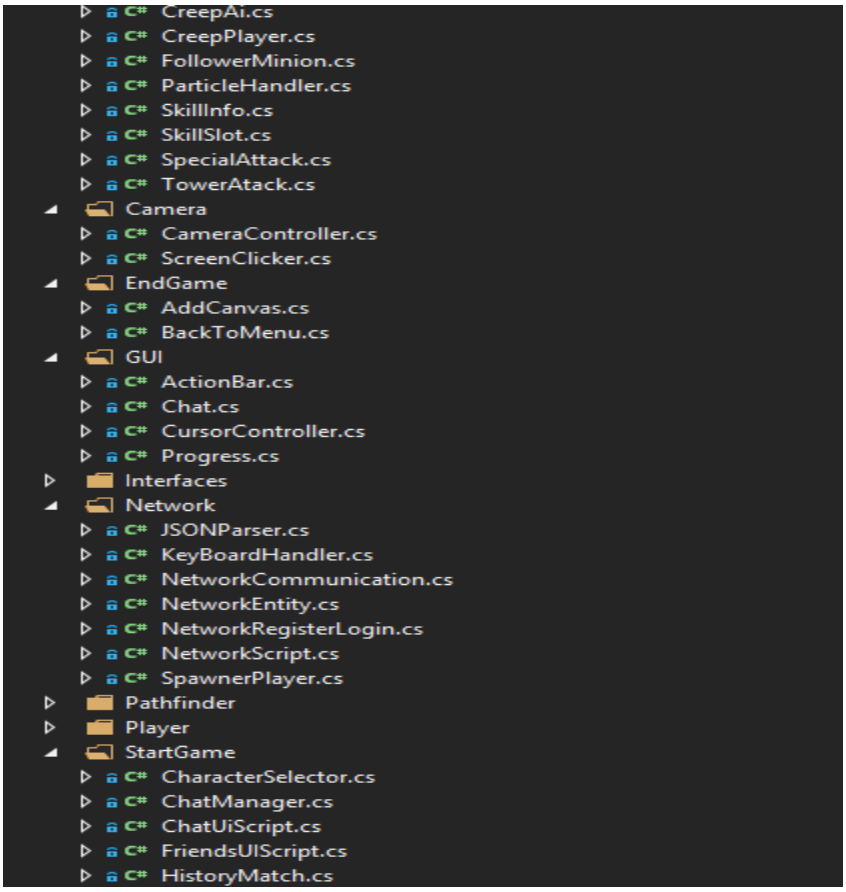


Figura 4: Modulele ce alcătuiesc clientul

1. Register - Înregistrarea unui nou utilizator prin completarea câmpurilor obligatorii.

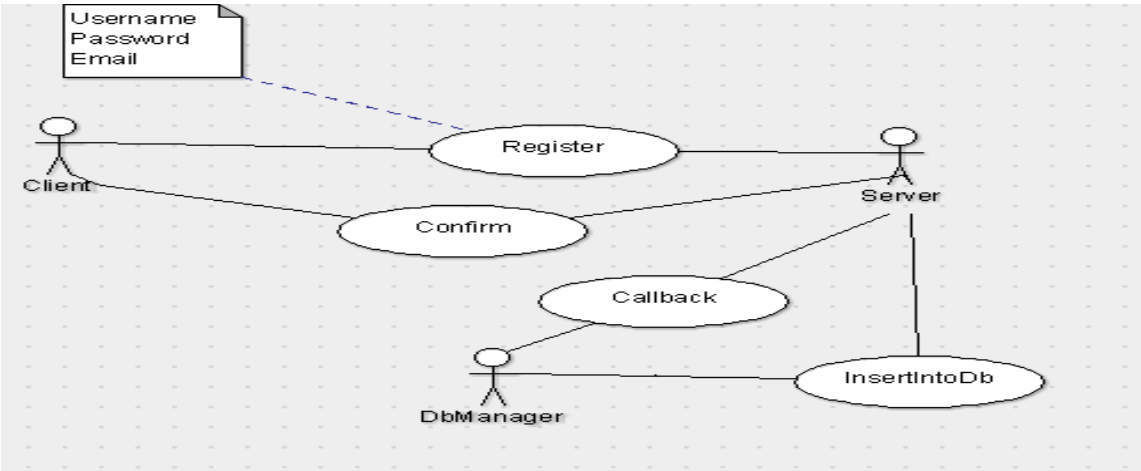


Figura 5: Diagramă înregistrare

2. Login - Autentificarea unui utilizator pe baza contului și parolei

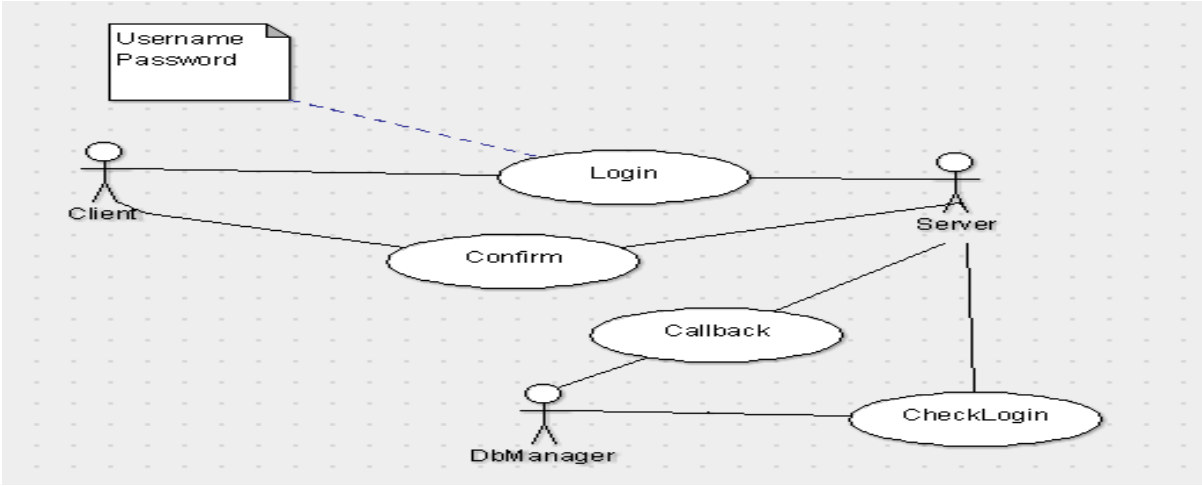


Figura 6: Diagramă autentificare

3. GameMenu - Crearea unei camere sau alăturarea la una deja existentă

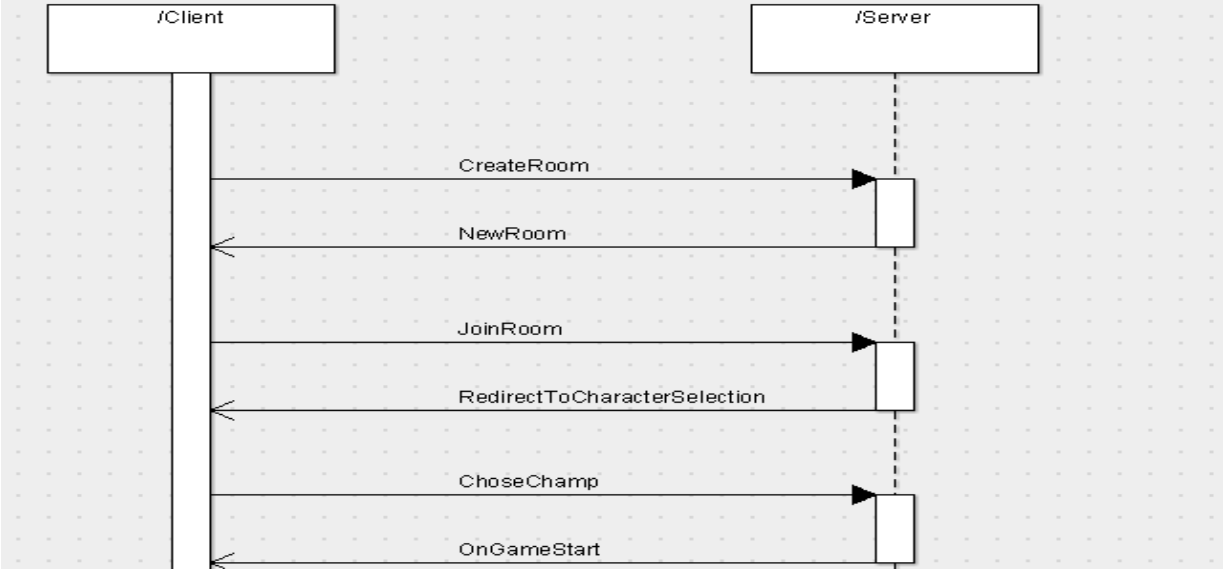


Figura 7: Diagramă creare/alăturare cameră

4. GameMenu - Posibilitatea adăugării unui prieten în listă sau eliminarea acestuia

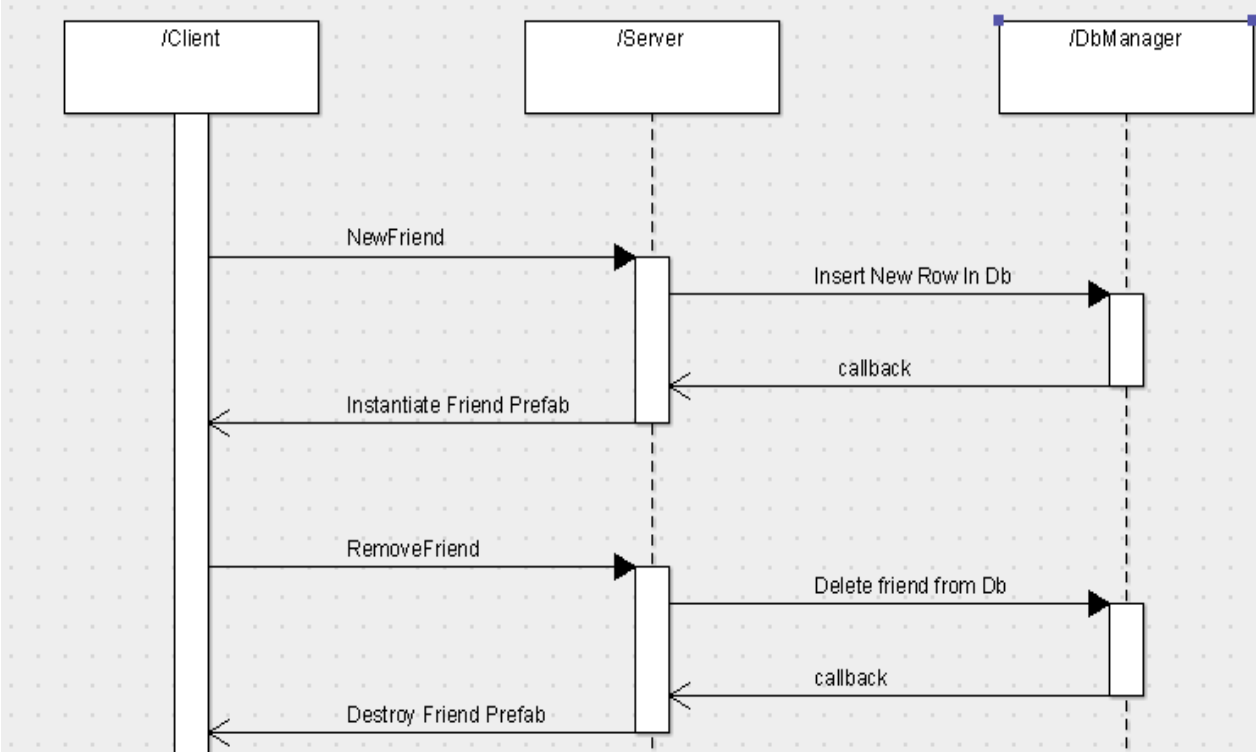


Figura 8: Diagramă adăugare/eliminare prieten

5. GameMenu - Modul de desfășurare al unei conversații între doi prieteni prin intermediul serverului. Lista de prieteni pentru cei doi clienți este actualizată și adusă din baza de date

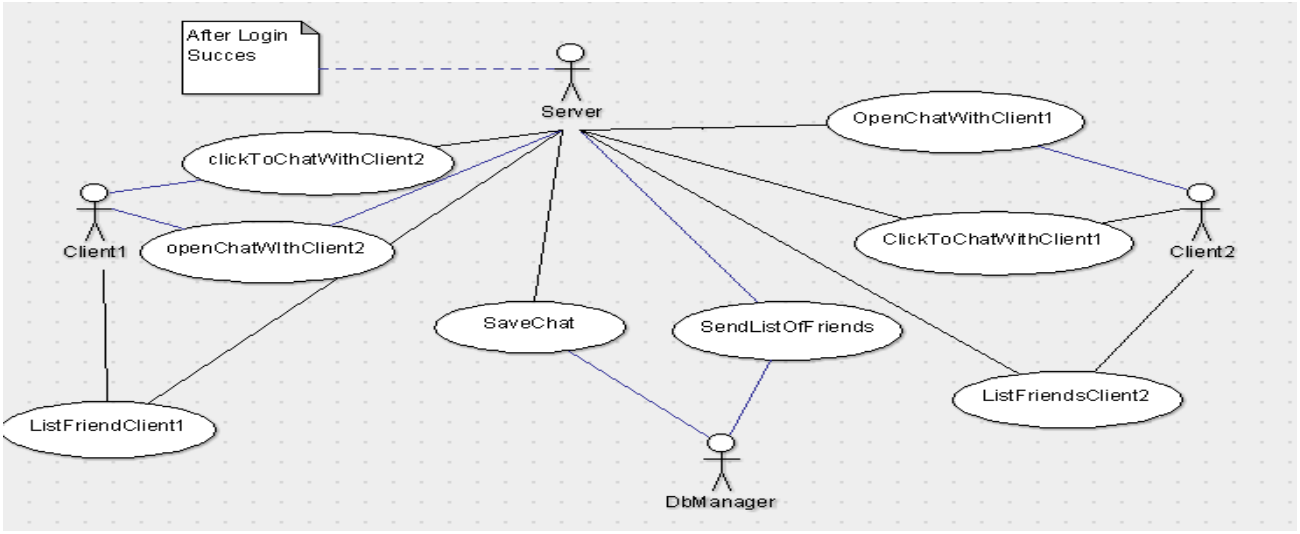


Figura 9: Diagrma chat în meniul principal

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

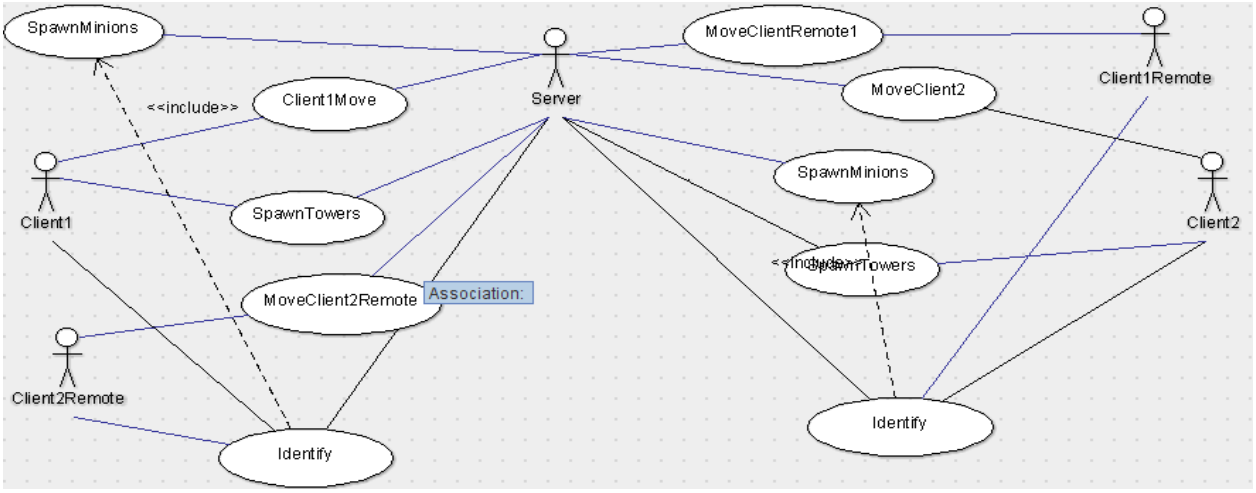


Figura 10: Diagramă pentru în joc pentru identificarea clienților

7. Game - Atac între utilizatori sincronizat prin intermediul serverului similar cu mișcarea acestora.

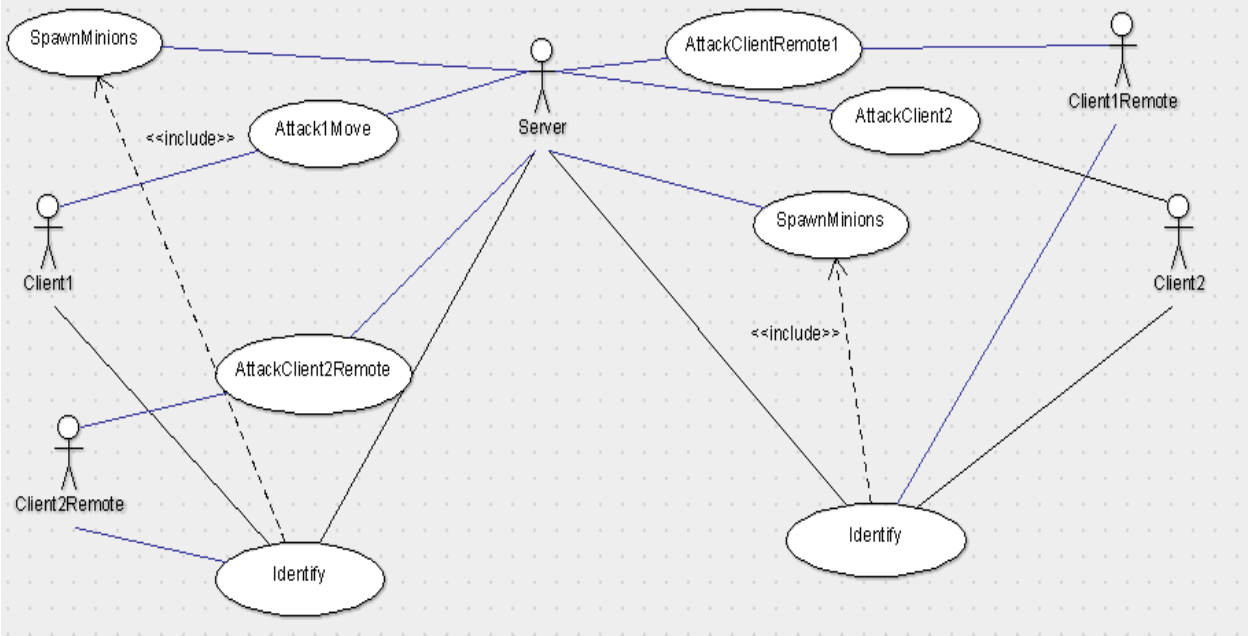


Figura 11: Diagramă atac utilizatori

8. Game - Urmărirea jucătorilor între ei, prin detectarea intrărilor asupra clonei celuilalt client. Evenimentul este trimis către server și apoi către clienți, în mod sincronizat.

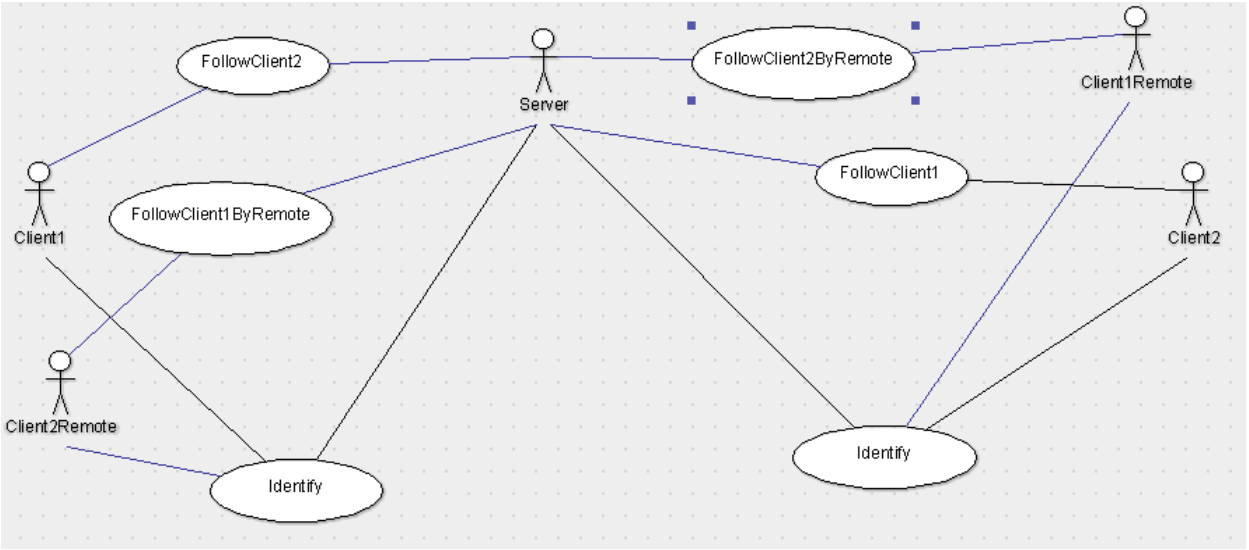


Figura 12: Diagramă urmărire clienți

9. Game - Mișcarea unităților și atacul acestora. Fiecare client are propriile unități și clonele unităților celuilalt client. Unitățile clientului clonă sunt controlate, la fel ca acesta, prin intermediul serverului.

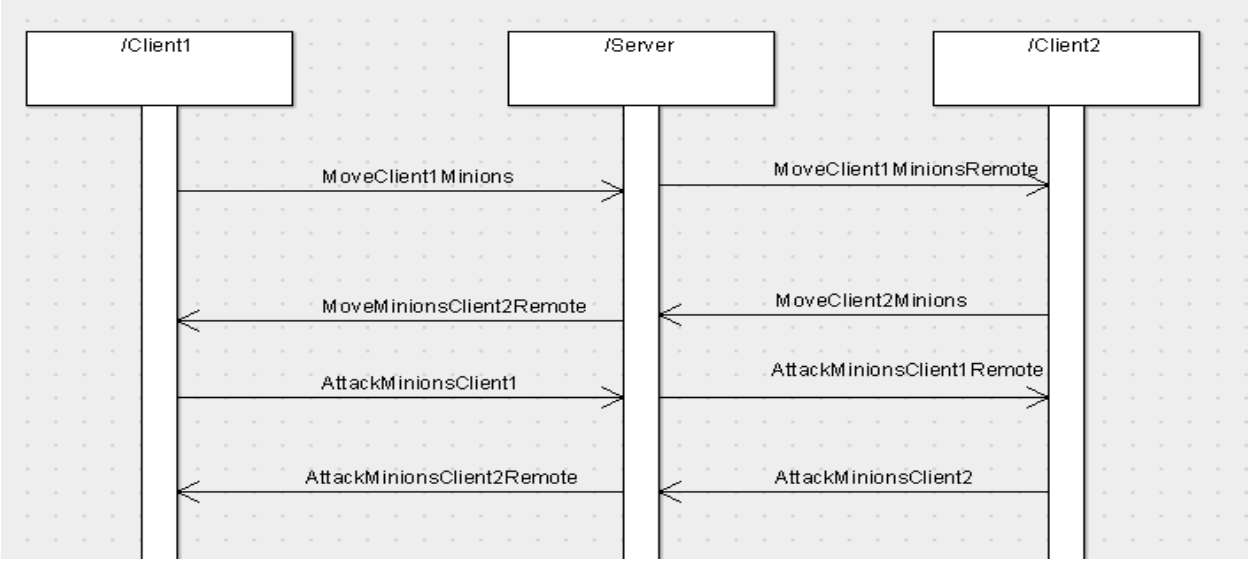


Figura 13: Diagramă deplasare și atac unități ajutătoare

10. Game - Urmărirea unităților și atacul acestora de către clienți. Este același principiu ca la unitățile campion controlate de client.

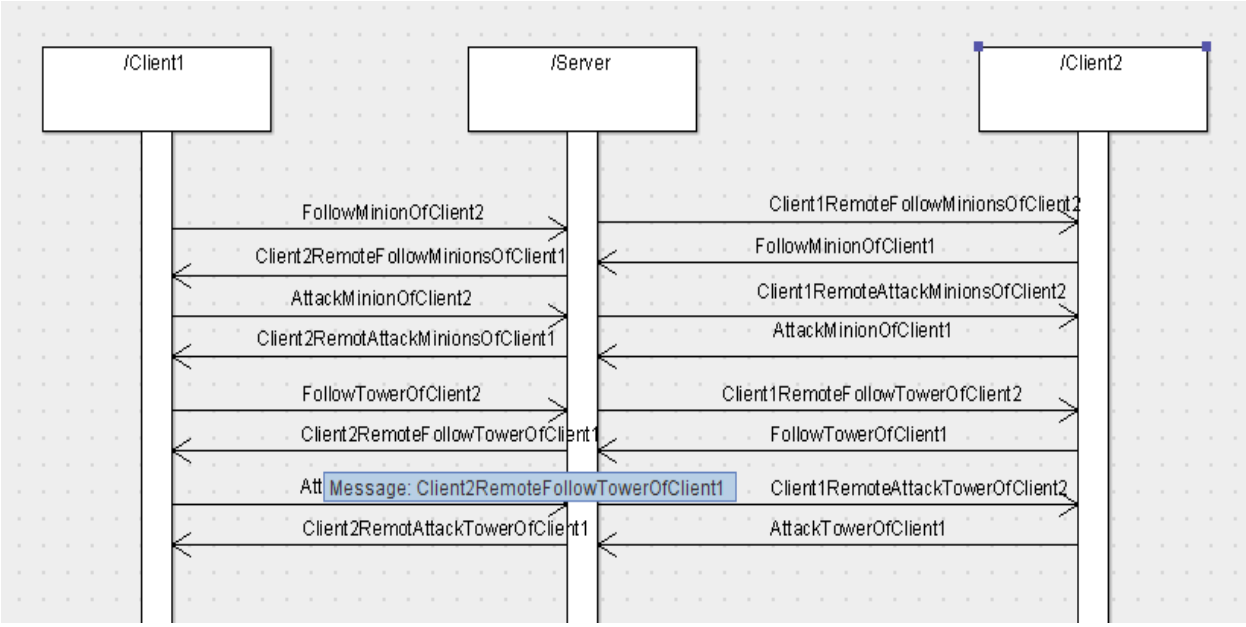


Figura 14: Diagramă interacțiune clienți unități adverse

11. Game - Urmărirea unităților și atacul între ele, dar și atacul asupra turnurilor.

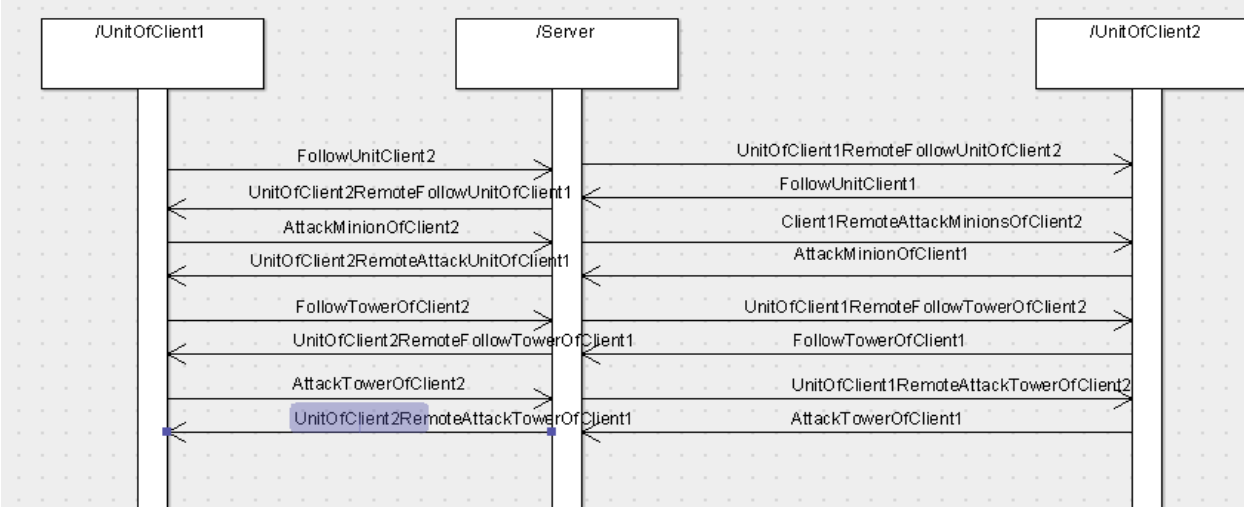


Figura 15: Diagramă de interacțiune a unităților și a turnurilor

12. Game - Conversațiile în joc se realizează doar în camera unde se desfășoră meciul

13.

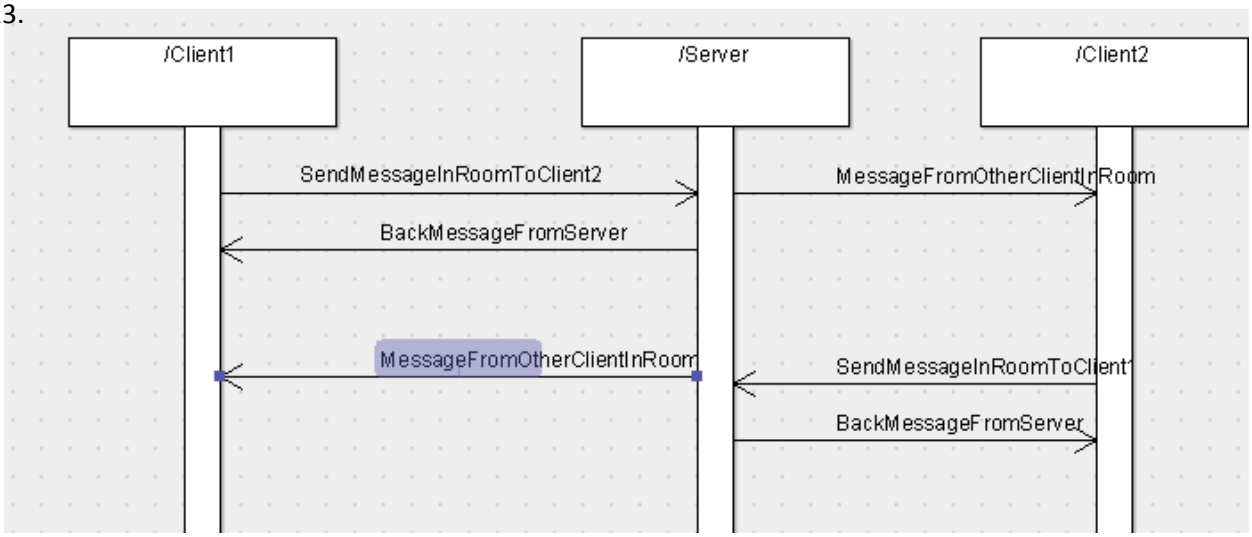


Figura 16: Diagramă chat în timpul unui joc

13.Game - Sfârsit joc si revenire către meniul principal atunci când un client distruge contrucția finală adversă.

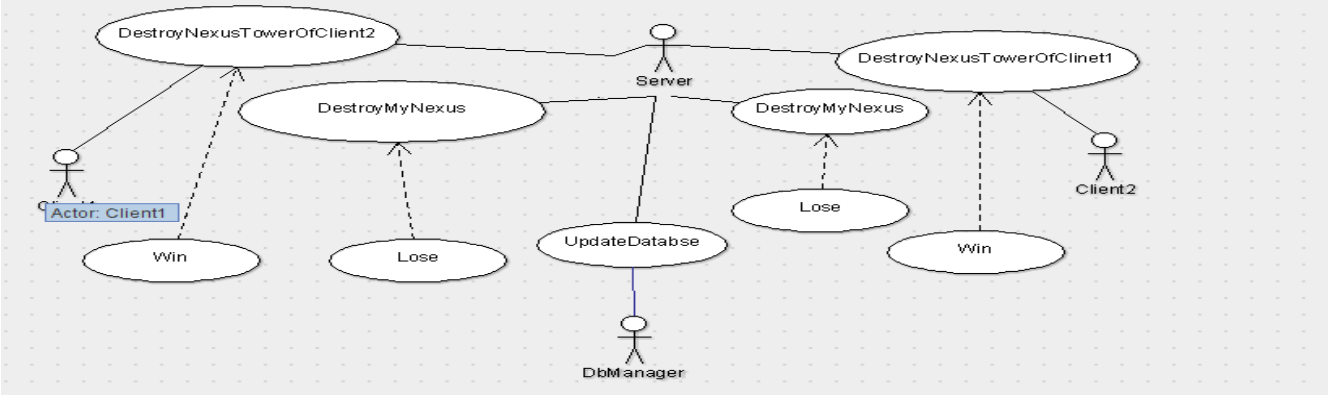


Figura 17: Diagrmă de sfârșit de joc

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

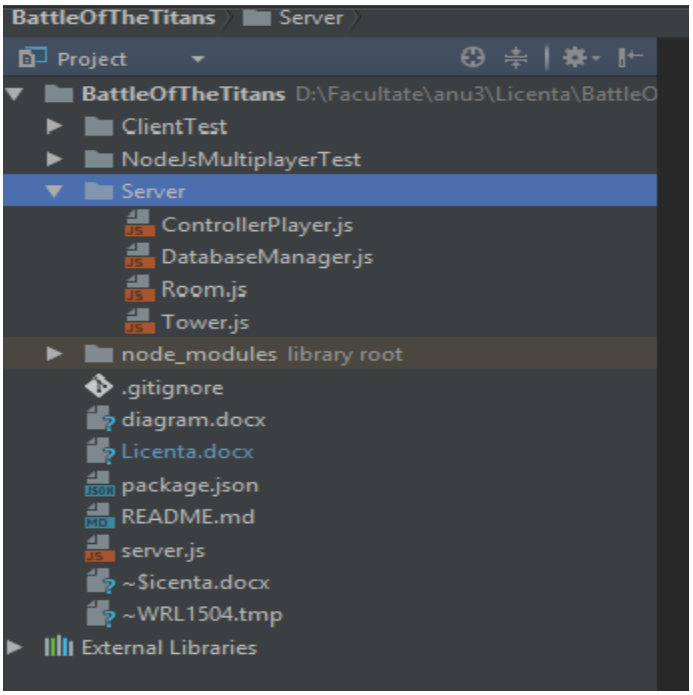


Figura 18 : Clasele ce alcătuiesc serverul

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, destul de flexibile, pentru crearea serverului de joc și susținerea meciurilor.

3.2 Modelarea datelor

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

3.2.1 Modelarea datelor pe partea de client

O noțiune foarte importantă în dezvoltarea unui joc prin intermediul game-engine-ului *Unity* este noțiune de *GameObject*[4], fiind cea mai abstractă clasă. Aceasta se află în spatele oricărui obiect tridimensional din 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 pentru a se obține 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*:

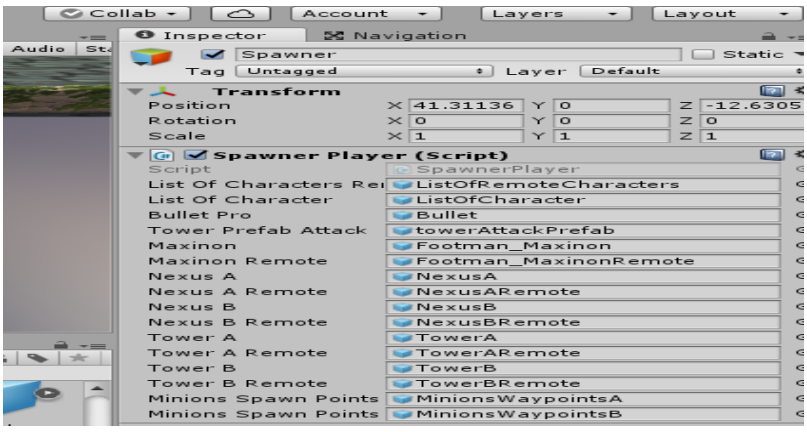


Figura 19: Componentele obiectului Spawner

[4] Clasa de bază pentru scenele din Unity https://docs.unity3d.com/ScriptReference/GameObject.html

Fiecare personaj din *ListOfCharacters* 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* conține tot ceea ce conține și lista principală de campioni, doar că la acesta se adăugă *FollowClick*, ce permite acțiuni asupra obiectelor în vederea urmăririi și atacării.

Footman_Maxion conține următoarele scripturi: *Target, PathFinder, NavigateToPosition, NetworkEntity, Alive, FollowMinions, CreepAi, CreepPlayer, AttackAI, FollowerMinion*. Pentru *Footman_MaxionRemote* s-a renuntat la *CreepPlayer*, dar s-au adăugat *CursorController* și *FollowMinions*.

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

3.2.2 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 alcătuită din 3 tabele principale.

Tabela *Users*:

```
CREATE TABLE `users` (
  `idUser` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `isOnline` tinyint(1) NOT NULL,
  PRIMARY KEY (`idUser`),
  UNIQUE KEY `idUser_UNIQUE` (`idUser`)
) ENGINE=InnoDB AUTO_INCREMENT=37 DEFAULT CHARSET=latin1
```

Figura 20: Tabelul Users din baza de date

Tabela *Friends*:

```
CREATE TABLE `friends` (
  `rowId` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `idUser` int(11) DEFAULT NULL,
  `idFriend` int(11) DEFAULT NULL,
  PRIMARY KEY (`rowId`)
) ENGINE=InnoDB AUTO_INCREMENT=56 DEFAULT CHARSET=latin1
```

Figura 21: Tabelul Friends din baza de date

Tabela *Statistics*:

```
CREATE TABLE `statistics` (
  `idstatistics` int(11) NOT NULL AUTO_INCREMENT,
  `idPlayer` int(11) NOT NULL,
  `namePlayer` varchar(45) NOT NULL,
  `oponentId` int(11) NOT NULL,
  `nameOponent` varchar(45) NOT NULL,
  `owner` tinyint(4) NOT NULL,
  `status` tinyint(4) NOT NULL,
  `totalDamage` int(11) NOT NULL,
  PRIMARY KEY (`idstatistics`)
) ENGINE=InnoDB AUTO_INCREMENT=28 DEFAULT CHARSET=latin1
```

Figura 22: Tabelul Statistics din baza de date

Câmpurile tabelelor sunt denumite în mod sugestiv. Pentru un utilizator, vom memora contul, parola și emailul. Vom folosi si un idUser care va permite identificarea mai ușoară atunci cand vom interoga și alte tabele. Pentru a aduce lista unui client se interogheză tabela *Friends* și se compară id-ul clientului cu câmpul *userId*. Tabela de statistici este folosită atunci când utilizatorul va accesa lista meciurilor desfășurate.

3.3 Protocoale de comunicare client – server

Biblioteca, care permite comunicarea bidirecțională în timp real între clienți și server, este Socket.io. Pe partea de client, este instalată componenta oferită de *Unity*, iar pe partea de server, cea oferită de *Node.js*.

Aceasta 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 scene.

Ideea principală din spatele Socket.IO este aceea că se pot trimite și primi evenimentele care se doresc, cu toate datele care se doresc. De asemenea se pot trimite orice obiecte care pot fi codificate ca *JSON*. Datele binare sunt suportate de asemenea.

Conectarea:

- la pornirea aplicației, pe partea de client:

```
private void Awake()
{
    SocketIO = GetComponent<SocketIOComponent>();
    if (created == false)
    {
        DontDestroyOnLoad(this.gameObject);
        SocketIO.Connect();
        created = true;
        SocketIO.On("allRooms", OnEnter);
        SocketIO.On("newRoom", OnNewRoom);
        SocketIO.On("closeRoom", OnCloseRoom);
        SocketIO.On("joinSuccesFull", OnJoinSucce
        SocketIO.On("roomFull", OnRoomFull);
        SocketIO.On("canPlay", OnCanPlay);
    }
    else
    {
        Destroy(this.gameObject);
    }
```

Figura 23: Conectarea clientului la server

  - 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:

```
server.js
var io = require('socket.io')(serv,{});


var playerNo = 0;
var roomNo = 0;
var ROOMS = {};

var mapingSocketRoom = {};
var globalPlayersLogged = {};
var mapNameInGameIdDatabase = {};
var idTowers = [];

io.sockets.on('connection', function(socket){
    console.log('Client connected is '+socket.id);
```

Figura 24: Așteptarea serverului pentru clienți

  - se obține modulul *socket.io* oferit de Node.js

- se așteptă conectarea clienților la un anumit port

Atunci când se dorește realizarea unui eveniment se transmit datele către server, numele evenimentului trimis de către client va trebui sa coincidă cu numele evenimentului pe partea de server. Astfel, evenimentul care se emite trebuie să aibă pereche pe partea de server. De exemplu atunci când se face autentificarea la sistem.

```
public void SendLoginData(string username, string password)
{
    SocketIO.Emit("login", new JSONObject(myJsonParser.LoginDataToJson(username, password)));
}
```

Figura 25: Exemplu de eveniment emis pe partea clientului

```
socket.on("register",onRegister);
socket.on("login",onLogin);
```

Figura 26: Exemplu de eveniment de tratat pe server

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. Fiecărui eveniment recepționat, atât pe partea de client, cât și pe partea de server, îi corespunde o funcție de tip callback.

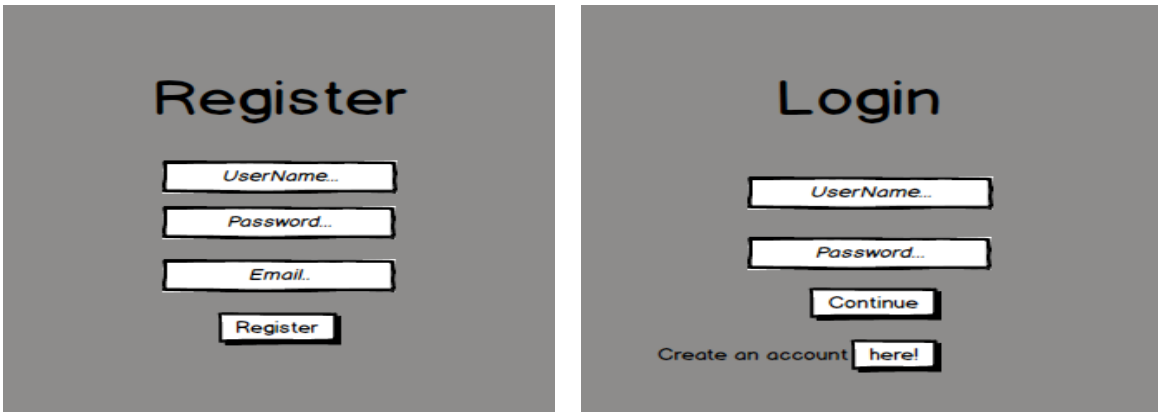
```
var onLogin = function(data){
    var socket = this;
    if(!checkAlreadyLog(data["username"]))
    {
        dbM.CheckLogin(data["username"],data["password"], functi
            if(err)
                console.log(err);

            if(err==="fail")
                socket.emit("wrongData");
            else if(err==="succes")
            {
                socket.emit("loginSuccesfull",{
                    username : data["username"]
                });


                globalPlayersLogged[socket.id] = data["username"
                //to do update databse for log
                //send array of connected frinds
                console.log("Dupa logare idu meu este "+id);
                mapNameInGameIdDatabase[data["username"]] = id;
                dbM.MakeLoginOnOff(data["username"],true);

                io.sockets.emit('updateListFriends');
```

Figura 27: Exemplu funcție callback

3.4 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 înțeles și de folosit, astfel încat utilizatorul să fie familiarizat cu elementele din care este alcătuită. Pentru fiecare scenă din joc, am realizat câte o interfața astfel:



Figurile 27 și 28: Interfață pentru înregistrare și autentificare

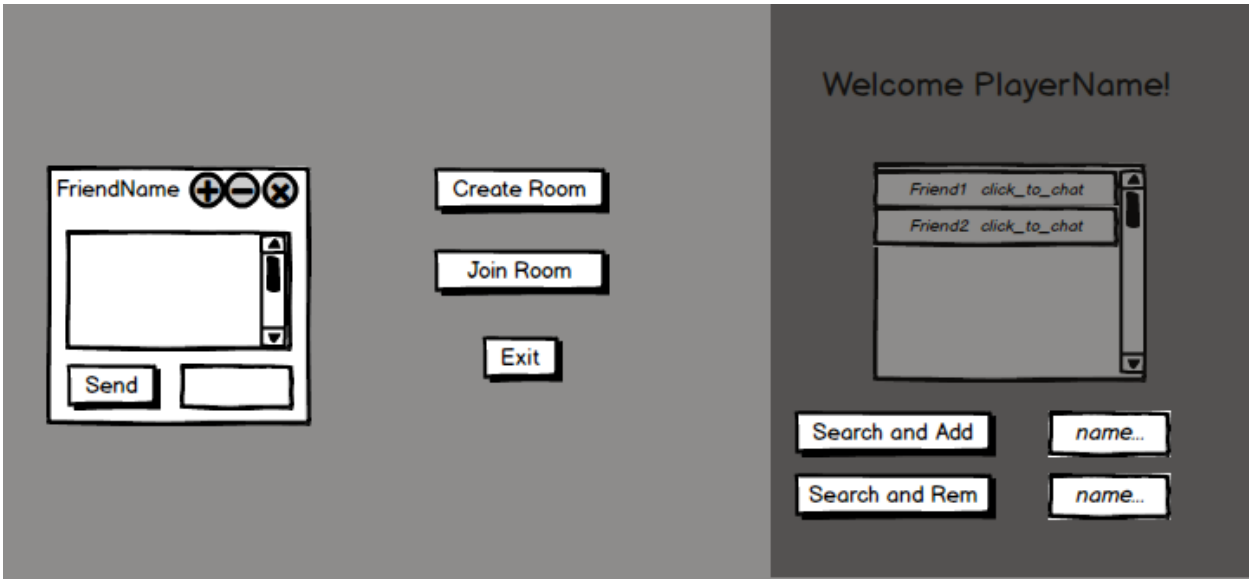


Figura 29: Intefață meniului principal

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 un buton ce va redirecționa utilizatorul către meniul de *Înregistrare*.

Acest meniu este similar cu precedentul. Se obervă ca cele două meniuri sunt simple și intuitive .

După autentificarea cu suces, utilizatorul va fi redirecționat către meniul jocului. Aici, utilizatorul 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 aplicației

În partea dreaptă, se află lista de prieteni ai utilizatorului. Acesta va putea porni o conversație cu utilizatorii activ. Printr-un simplu click asupra unui membru din lista de prieteni, se va deschide o casetă de conversație, unde va avea loc discuția. Acestă casetă este prevazută cu 3 butone, care vor acționa asupra ei, realizând unul dintre evenimentele de micșorare, de mărire sau de închidere.

Când camera va fi ocupată la potențialul maxim de utilizatori se va porni scena de alegere a unităților campion. După confimarea celor două personaje se va porni scena jocului.

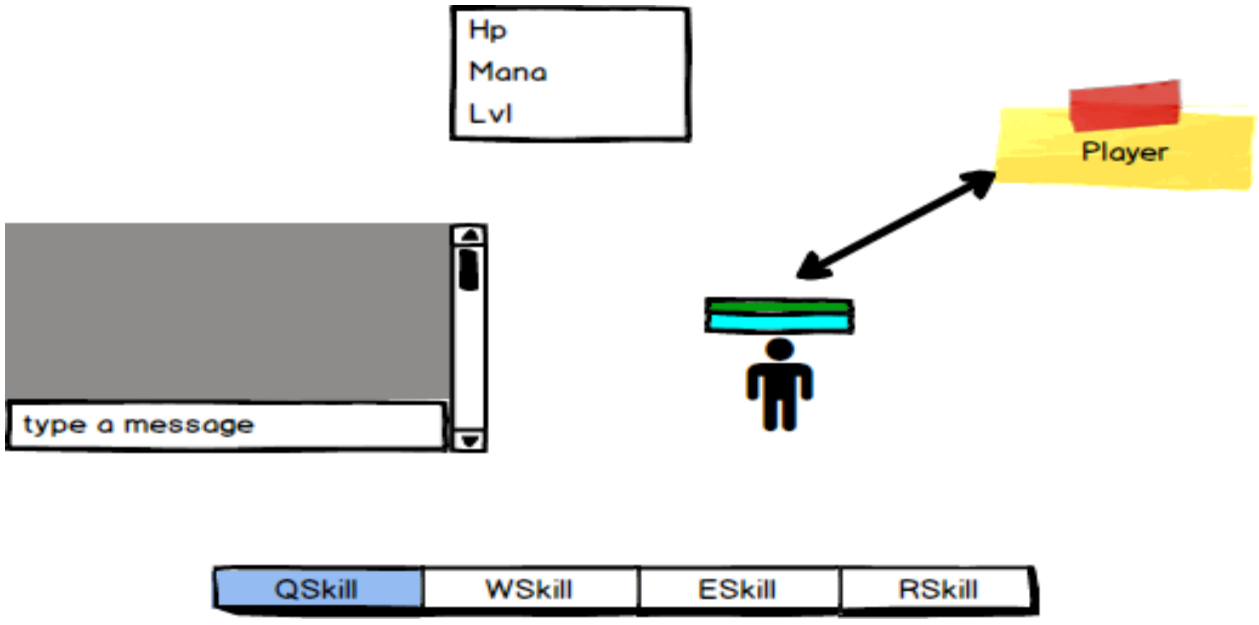


Figura 30: Interfața joc

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

4 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
- Aplicarea algoritmului A* Pathfinding optimizat pe structuri de heap
- Proiectarea unui alogritm de inteligență artificială pentru controlul unităților ajutătoare
- Sistemul de atac magic
- Dezvoltarea serverului semi-autoritar

4.1 Chat

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

```
var DatabaseManager = function(){
    var self = this;
    this.mysql      = require('mysql');
    this.connection = this.mysql.createConnection({
        host     : 'localhost',
        user     : 'root',
        password : '',
        database : 'battleoftitans'
    });

    this.connection.connect();
    this.InserIntoUsers = function(_username, _password, _email, cb){...
    }
    this.CheckLogin = function(_username, _password, cb){...
    }
    this.MakeLoginOnOff = function(_username, status){
        this.connection.query('UPDATE users SET isOnline = ? WHERE username = ?', [status, _username]);
    }
    this.InsertFriend = function(myId, nameFriend, cb){...
    }
    this.RemoveFriend = function(myId, nameFriend, cb){...
    }
    this.GetListOfFriendById = function(myId, cb){...
    }
    this.GetIdFromUserByName = function(nameUser, callback){...
    }

}
```

Figura 31: Clasa DatabaseManager

După autentificarea cu succes a clientului, se va cere de la server lista de prieteni pe care acesta îi deține. Acesta listă este obținută prin interogarea bazei de date către tabela *Friends*. Totodată când un anumit client se autentifică, este trimisă către toti clienții lista pentru actualizare, în cazul în 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 căruia i-a fost adresată. Clientul va fi notificat, pe ecran se va deschide o casetă de dialog, cu mesajul corespunzător.

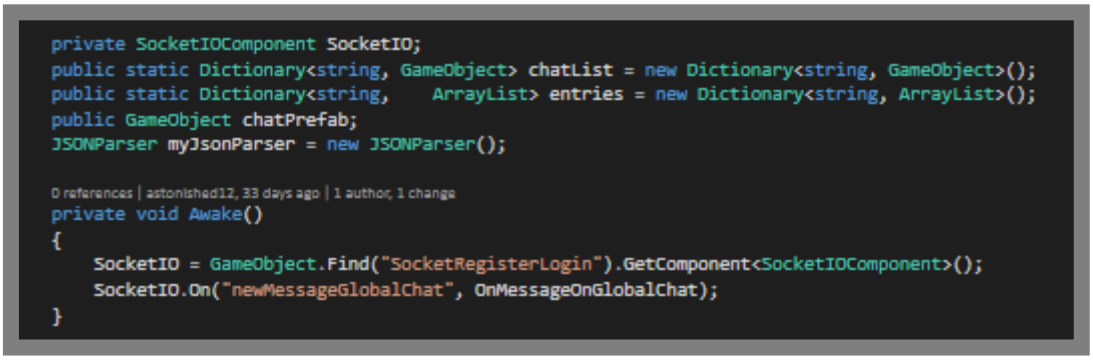
Astfel pe fiecare client se construiește un nou obiect de tipul **friendPrefab**, obiect ce va fi adăugat in lista prietenilor. Lista prietenilor va fi compusă din prietenii care sunt activi, dar și din cei care nu sunt activi. Schimbul de mesaje între clienți se poate realiza numai daca aceștia sunt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

ChatManager

Pentru a reține activitatea pe care utilizatorul o va desfășură împreuna cu prietenii acestuia, am construit o clasă ce va avea ca principal scop monitorizarea schimburilor de mesaje și a discuțiilor.

Astfel daca o persoană vrea să trimită un mesaj către un alt client, în momentul recepționării se va construi un obiect de tipul **ChatUI**, ce va fi creat doar odată. Nu vor exista două 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.



```
private SocketIOComponent SocketIO;
public static Dictionary<string, GameObject> chatList = new Dictionary<string, GameObject>();
public static Dictionary<string,    ArrayList> entries = new Dictionary<string, ArrayList>();
public GameObject chatPrefab;
JSONParser myJsonParser = new JSONParser();

0 references | astonished12, 33 days ago | 1 author, 1 change
private void Awake()
{
    SocketIO = GameObject.Find("SocketRegisterLogin").GetComponent<SocketIOComponent>();
    SocketIO.On("newMessageGlobalChat", OnMessageOnGlobalChat);
}
```

Figura 32a: Clasa ChatManager

```
1 reference | astonished12, 10 days ago | 1 author, 4 changes
private void OnMessageOnGlobalChat(SocketIOEvent Obj)
{
    string socket_id = myJsonParser.ElementFromJsonToString(Obj.data["socket_id"].ToString())[1];
    string listener = myJsonParser.ElementFromJsonToString(Obj.data["name"].ToString())[1];
    string message = myJsonParser.ElementFromJsonToString(Obj.data["message"].ToString())[1];
    if (!chatList.ContainsKey(listener)){
        GameObject chat = Instantiate(chatPrefab);
        chat.transform.FindChild("To").GetComponent<Text>().text = listener;
        chat.transform.SetParent(gameObject.transform, false);
        chatList.Add(listener, chat);
    }
    chatList[listener].GetComponent<ChatUiScript>().inputField = message;

    chatList[listener].GetComponent<ChatUiScript>().senderId = socket_id;
    if (chatList[listener].GetComponent<ChatUiScript>().newMessage)
    {
        chatList[listener].GetComponent<ChatUiScript>().AddChatEntry(NetworkRegisterLogin.UserName, chatList[listener].GetComponent<ChatUiScript>().inputField, true);
    }
    else
    {
        chatList[listener].GetComponent<ChatUiScript>().AddChatEntry(listener, chatList[listener].GetComponent<ChatUiScript>().inputField, false);
    }
    chatList[listener].GetComponent<ChatUiScript>().newMessage = false;

}
```

Figura 32b: Clasa ChatManager

Conversațiile nu vor fi salvate în baza de date deoarece chatul a fost proiectat în scop informativ pentru a fi suport în legătura cu jocul.

4.2 Sistemul de camere

Pentru a disputa o partidă, clientul fie creează o cameră nouă în care va astepta ca un alt utilizator să 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 cameră. Serverul va prelua acest semnal și va stoca acest lucru.

```
createRoomButton.onClick.AddListener(() => {
    menu.SetActive(false);
    joinMenu.SetActive(false);
    createMenu.SetActive(true);
    SocketIO.Emit("newRoom");
});
```

Figura 33: Funcția butonului newRoom

```
var  Room = function (id,name, maxPlayers) {
        //this.io = io;
        this.id = id;
        this.maxPlayers = maxPlayers;
        this.name = name;
        this.currentPlayers = 1;
        this.PLAYERS = {};
        this.confirmedCharacters = 0;
        this.towersId = [];
        this.minionsId = [];
        this.lastTimeSpawn = -999;
        this.spawnCheck = false;
        this.towers = {};
        this.minions = {};
    };
```

Figura 34: Clasa Room pe server

Se va seta deținătorul camerei, se va memora numele camerei, se vor genera identificatorii turnurilor folosindu-se de modulul **shortid** oferit de N*ode.js*. Camera se va memora in dicționarul global **ROOMS**, memorare ce va avea ca și cheie socket-id-ul utilizatorului.

Clientul va fi identificat cu socketul înca de la conectarea la server. Folosind acest lucru, am alăturat socketul cu camera proaspăt creată. De asemenea, se va construi și un obiect de tip **ControllerPlayer**, ce va fi adăugat in lista jucătorilor camerei. Acest lucru va susține autoritatea serverului.

```
this.join(ROOMS[this.id].name);
```

Figura 35: Alăturarea socketului la cameră

```
this.broadcast.emit("newRoom",{
    socket_id : ROOMS[this.id].id,
    maxPlayers : ROOMS[this.id].maxPlayers,
    currentPlayers : ROOMS[this.id].currentPlayers,
    name : ROOMS[this.id].name
});
```

Figura 36: Anunțarea utilizatorilor - newRoom

Serverul va trimite către toți utilizatorii conectați faptul că s-a deschis o nouă cameră, trimițandu-le datele acesteia.

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

```
public static Dictionary<string,Room> RoomList = new Dictionary<string,Room>();
```

Figura 37: Dicționarul camerelor pe client

Dacă se dorește alăturarea la o anumită cameră se va transmite către server evenimentul de **joinRoom,** se va actualiza camera respectivă, incrementându-se numarul jucătorilor. Socketul clientului se va alătura la cameră, după care severul va trimite catre cei doi participanți evenimentul **joinSuccesFull**.

```
var onJoinRoom = function(data){
    if(ROOMS[data["idRoom"]].currentPlayers >= ROOMS[data["idRoom"]].maxPlayers)
    {
        this.emit("roomIsFull");
    }
    else
    {
        io.sockets.emit("roomFull",{
            room_id : data["idRoom"]});

        ROOMS[data["idRoom"]].currentPlayers++;
        var player = new ControllerPlayer(this.id,globalPlayersLogged[this.id],88,0,-5,"false");
        ROOMS[data["idRoom"]].PLAYERS[this.id] = player;
        //console.log(ROOMS[data["idRoom"]].PLAYERS);
        mapingSocketRoom[this.id] = ROOMS[data["idRoom"]];
        this.join(ROOMS[data["idRoom"]].name);

        io.to(ROOMS[data["idRoom"]].name).emit("joinSuccesFull",{
            room_id :  data["idRoom"]});

    }

}
```

Figura 38: Funcția onJoinRoom pe server

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

4.3 Algoritmul A* Pathfinding

4.3.1 Introducere

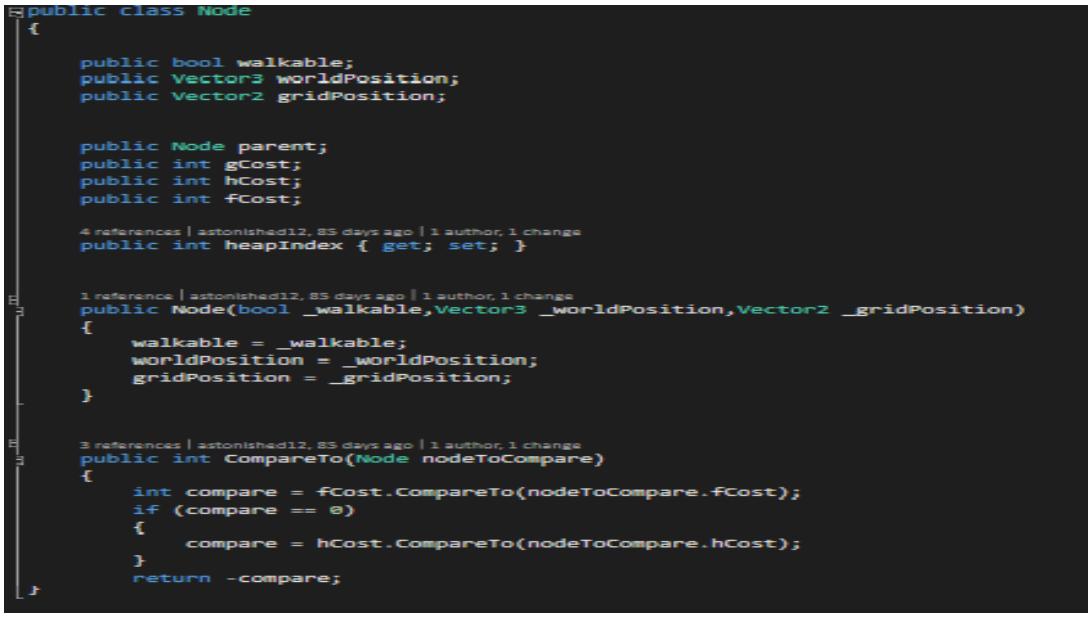
Am folosit acest algoritm pentru a realiza mișcarea personajului principal în timpul jocului. Algoritmul A* este unul dintre cei mai populari algoritmi pentru cautarea traseelor.

Algoritmul oferă cel mai scurtt 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 inițial. 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 soluție dacă acesta există
- poate folosi o strategie care să mărească viteza de executare a procesului de căutare
- nodurile pot avea diferite costuri, există posibilitatea ca unele drumuri să fie mai dificil de traversat.
- dacă se dorește, algoritmul poate căuta în mai multe direcții (3)

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



```
public class Node
{

    public bool walkable;
    public Vector3 worldPosition;
    public Vector2 gridPosition;


    public Node parent;
    public int gCost;
    public int hCost;
    public int fCost;

    4 references | astonished12, 85 days ago | 1 author, 1 change
    public int heapIndex { get; set; }


    1 reference | astonished12, 85 days ago | 1 author, 1 change
    public Node(bool _walkable,Vector3 _worldPosition,Vector2 _gridPosition)
    {
        walkable = _walkable;
        worldPosition = _worldPosition;
        gridPosition = _gridPosition;
    }


    3 references | astonished12, 85 days ago | 1 author, 1 change
    public int CompareTo(Node nodeToCompare)
    {
        int compare = fCost.CompareTo(nodeToCompare.fCost);
        if (compare == 0)
        {
            compare = hCost.CompareTo(nodeToCompare.hCost);
        }
        return -compare;
}
```

Figura 39: Clasa Node

Pentru fiecare nod în parte, trebuie stabilite 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. Este cel mai important cost deoarece influențează modul de deplasare.
- **costul F**: este costul total al drumului 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 euristici sunt: Distanța Manhattan, Distanța Euclidiană, Distanța Diagonală având cost uniform, Distanța Diagonală fară cost uniform. Algoritmul implementat în cadrul jocului BattleOfTitans folosește ca euristică Distanța Diagonală, ce va avea costul pe diagonală 14, iar pentru restul direcțiilor 10. (4)

```
internal int GetHeuristicDistance(Node start, Node target)
{
    //x10 work with int

    int distanceOfXCoordinate = (int)Mathf.Abs(start.gridPosition.x - target.gridPosition.x);
    int distanceOfYCoordinate = (int)Mathf.Abs(start.gridPosition.y - target.gridPosition.y);

    if (distanceOfXCoordinate > distanceOfYCoordinate)
        return costOfDiagonalMovement * distanceOfYCoordinate + 10 * (distanceOfXCoordinate - distanceOfYCoordinate);

    return costOfDiagonalMovement * distanceOfXCoordinate + costOfNonDiagonalMovement * (distanceOfYCoordinate - distanceOfXCoordinate);
}
```

Figura 40: Distanța diagonală

Astfel, harta va avea ca și componentă principală scriptul **Grid** unde se va realiza împartirea în noduri. Se calculează punctul aflat în 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 acesteia. Se construiește un obiect de tipul **Node** și se adaugă la obiectul matricial.

```
1 reference | astonished12, 85 days ago | 1 author, 1 change
void MakeGrid()
{
    gridTable = new Node[gridSizeX, gridSizeY];
    /// 0,0,0 - 1,0,0* max(x)/2 - 0,0,1 * max(y)
    Vector3 worldBottomLeft = transform.position - new Vector3(1,0,0) * gridWorldSize.x / 2 - new Vector3(0,0,1) * gridWorldSize.y / 2;

     for (int x = 0; x < gridSizeX; x++)
     {
         for (int y = 0; y < gridSizeY; y++)
         {
            Vector3 pointInWorldMap = worldBottomLeft + new Vector3(1, 0, 0) * (x+0.5f) + new Vector3(0, 0, 1) * (y+0.5f);
            bool checkIsWalkble = !(Physics.CheckSphere(pointInWorldMap, 0.5f, unwalkableMask));
            gridTable[x, y] = new Node(checkIsWalkble, pointInWorldMap, new Vector2(x,y));
        }
     }
}
```

Figura 41: Funcția care construiește matricea

Având acest obiect matricial, putem verifica de fiecare dată ce se află pe hartă la anumite coordonate. Pentru a stabili destinația unde trebuie să ajungă personajul, calculam indecsii 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 proprietatea de accesare, acesta se va utiliza ca destinație finală pentru alogritmul A*. În cazul contrar, este returnat primul vecin al acestuia, având cea mai apropiată distanță. Acest lucru este menținut de structura *Heap* pe care se bazează fiecare cautare.

```
3 references | astonished12, 33 days ago | 1 author, 2 changes
public Node GetNodeFromWorldPoint(Vector3 positionOnWorldMap)
{
    float percentX = (positionOnWorldMap.x + gridWorldSize.x / 2) / gridWorldSize.x;
    //3d to 2d z = y . Y nu se modifica;
    float percentY = (positionOnWorldMap.z + gridWorldSize.y / 2) / gridWorldSize.y;
    percentX = Mathf.Clamp01(percentX);
    percentY = Mathf.Clamp01(percentY);

    int x = Mathf.RoundToInt((gridSizeX - 1) * percentX);
    int y = Mathf.RoundToInt((gridSizeY - 1) * percentY);

    if (gridTable[x, y].walkable)
        return gridTable[x, y];
    else
        return GetNeighborsByNode(gridTable[x, y])[0];
}
```

Figura 42: Obținerea nodului pe baza coordonatelor

Pseudocod

```
function A*(start, goal)
  open_list = set containing start
  closed_list = empty set
  start.g = 0
  start.f = start.g + heuristic(start, goal)
  while open_list is not empty
    current = open_list element with lowest f cost
    if current = goal
      return construct_path(goal) // path found
    remove current from open_list
    add current to closed_list
    for each neighbor in neighbors(current)
      if neighbor not in closed_list
        neighbor.f = neighbor.g + heuristic(neighbor, goal)
        if neighbor is not in open_list
          add neighbor to open_list
        else
          openneighbor = neighbor in open_list
          if neighbor.g < openneighbor.g
            openneighbor.g = neighbor.g
            openneighbor.parent = neighbor.parent
  return false // no path exists

function neighbors(node)
  neighbors = set of valid neighbors to node // check for obstacles here
  for each neighbor in neighbors
    if neighbor is diagonal
      neighbor.g = node.g + diagonal_cost // eg. 1.414 (pythagoras)
    else
      neighbor.g = node.g + normal_cost // eg. 1
    neighbor.parent = node
  return neighbors

function construct_path(node)
  path = set containing node
  while node.parent exists
    node = node.parent
    add node to path
  return path
```

Figura 43: Pseudocode A* Pathfinding (5)

4.3.2 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. Acestă structură oferă posibilitatea stocării elementelor intr-un vector.
- Structura poate fi de tipul **MinHeap** sau **MaxHeap**. Într-o structură 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, am folosit o structură de tipul *MinHeap* în care se rețin nodurile cele mai apropiate față de punctul de căutare.

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

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

- Introducerea unui nou element în 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 pot determina încălcarea proprietăților de heap. (6)

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.

```
1 reference | astonished12, 93 days ago | 1 author, 1 change
private void GoUp(int p)
{
    while (p > 1 && heapArray[p].CompareTo(heapArray[p / 2])<0)
    {
        Swap(p, p / 2);
        p /= 2;
    }
}
```

Figura 44: Funcția GoUp

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.

```
private void Swap(int p, int q)
{
    Node aux = heapArray[p];
    heapArray[p] = heapArray[q];
    heapArray[q] = aux;
    heapArray[p].heapIndex = p;
    heapArray[q].heapIndex = q;
}
```

Figura 45: Interschimbarea nodurilor în heap

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

```
private void GoDown(int p)
{
    int fiust = 2 * p, fiudr = 2 * p + 1, bun = p;

    if (fiust <= currentSize && heapArray[fiust].CompareTo(heapArray[bun])<0)
        bun = fiust;
    // determin fiul "minim"
    if (fiudr <= currentSize && heapArray[fiudr].CompareTo(heapArray[bun])<0)
        bun = fiudr;

    if (bun != p)
    {
        Swap(p, bun);
        GoDown(bun);
    }
}
```

Figura 46: Funcția GoDown

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

Tabelă complexități

Data structure	Insert	Extract minimum	Decrease key
Array	$\Theta(1)^*$	$O(n)$	$\Theta(1)$
Sorted array	$O(\log n)$	$\Theta(1)$	$O(n)$
Binary heap	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
Fibonacci heap	$\Theta(1)$	$O(\log n)^*$	$\Theta(1)^*$

Figura 47: Tabela complexități operații pe heap

4.3.3 Aplicare algoritm

Algoritmul presupune crearea a două liste:

- lista cu nodurile nevizitate (en. Open List)
- lista cu nodurile vizitate (en. Closed List).

Prima listă se va utiliza pentru a reține cele mai bune noduri candidat dintre nodurile care nu au fost luate în considerare până în acel moment pentru traversare. Se va începe cu nodul de start. Dacă lista rămâne goală însemnă că 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 nodurilor nevizitate. Dacă acesta nu este nodul destinație, atunci algoritmul îi adaugă toți vecinii valizi în lista cu noduri nevizitate și repetă procesul.

```
public class PathFinder : MonoBehaviour {
    private Grid gridTable;
    0 references | astonished12, 67 days ago | 1 author, 1 change
    private void Awake(){
        GameObject ground = GameObject.Find("Ground");
        gridTable = ground.GetComponent<Grid>();
    }
    2 references | astonished12, 93 days ago | 1 author, 1 change
    public List<Node> AStar(Vector3 startPosition, Vector3 targetPosition){
        gridTable.ResetGridTable();
        Node start = gridTable.GetNodeFromWorldPoint(startPosition);
        Node target = gridTable.GetNodeFromWorldPoint(targetPosition);
        Heap openList = new Heap(1000);
        HashSet<Node> closedList = new HashSet<Node>();
        openList.Insert(start);
        start.gCost = 0;
        start.hCost = gridTable.GetHeuristicDistance(start, target);
        start.fCost = start.gCost + start.hCost;
        while (!openList.IsEmpty()) {
            Node currentNode = openList.GetRoot();
            if (currentNode.Equals(target)) {

                return ConstructPath(currentNode);
            }
             closedList.Add(currentNode);
            List<Node> neighbors = gridTable.GetNeighborsByNode(currentNode);
            foreach (Node neighbour in neighbors)
            {
                if (closedList.Contains(neighbour))
                    continue;
                int newMovementCostToNeighbour = currentNode.gCost + gridTable.GetHeuristicDistance(currentNode, neighbour);
                if (newMovementCostToNeighbour < neighbour.gCost || !openList.Contains(neighbour))
                {
                    neighbour.gCost = newMovementCostToNeighbour;
                    neighbour.hCost = gridTable.GetHeuristicDistance(neighbour, target);
                    neighbour.parent = currentNode;
                    if (!openList.Contains(neighbour))
                        openList.Insert(neighbour);
                }
            }
        }
        return new List<Node>();
    }
```

Figura 48: Clasa PathFinder

Clasa *PathFinder* va fi folosită pentru a determina care este drumul minim între poziția unde se află personajul 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 atunci va fi apelată metoda **ConstructPath**.

```
public List<Node> ConstructPath(Node node)
{
    List<Node> path = new List<Node>();
    path.Add(node);
    while (node.parent != null)
    {
        path.Add(node);
        node = node.parent;
    }
     path.Reverse();
    gridTable.path = path;
    return path;
}
```

Figura 49: Funcția care construiește drumul minim

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.

```
3 references | astonished12, 31 days ago | 1 author, 7 changes
public void SetTargetPosition(Vector3 _targetPosition)
{
    currentTargetPosition = _targetPosition;
    var pathfinder = GetComponent<PathFinder>();
    path = pathfinder.ASta
    SetDestination(path);
    target.SetTargetTransf
    animator.SetBool("attack", false);

}
```

PathFinder Component.GetComponent<PathFinder>()

Figura 50: Funcția cu setează posibila locație

În cadrul funcției *SetDestination,* se va declanșa o **corutină** ce va permite orientarea și deplasarea unității către ultimul nod din lista drumului calculat.

O corutină este o funcție ce are capacitatea de a întrerupe execuția programului principal și de a oferi control înapoi către Unity. De asemenea se poate ca la urmatorul cadru să reia execuția de unde a rămas.

```
public void SetDestination(List<Node> _path){
    targetSuccesfull = false;
    GetComponent<Animator>().SetBool("atDestination", false);
    path = _path;
    targetIndex = 0;
    StopCoroutine("FollowPath");
    StartCoroutine("FollowPath");
}
0 references | astonished12, 82 days ago | 1 author, 6 changes
IEnumerator FollowPath(){
    Vector3 currentWaypoint = path[0].worldPosition;
    while (true)
    {
        if (transform.position == currentWaypoint)
        {
            targetIndex++;
            if (targetIndex >= path.Count - 1)
            {
                targetSuccesfull = true;
                yield break;
            }
            currentWaypoint = path[targetIndex].worldPosition;

        }
        transform.rotation = Quaternion.LookRotation(currentWaypoint - transform.position);
        transform.position = Vector3.MoveTowards(transform.position, currentWaypoint, speed*Time.deltaTime);
        yield return null;
    }
}
```

Figura 51: Funcția SetDestination și corutina FollowPath

Animatorul unității campion

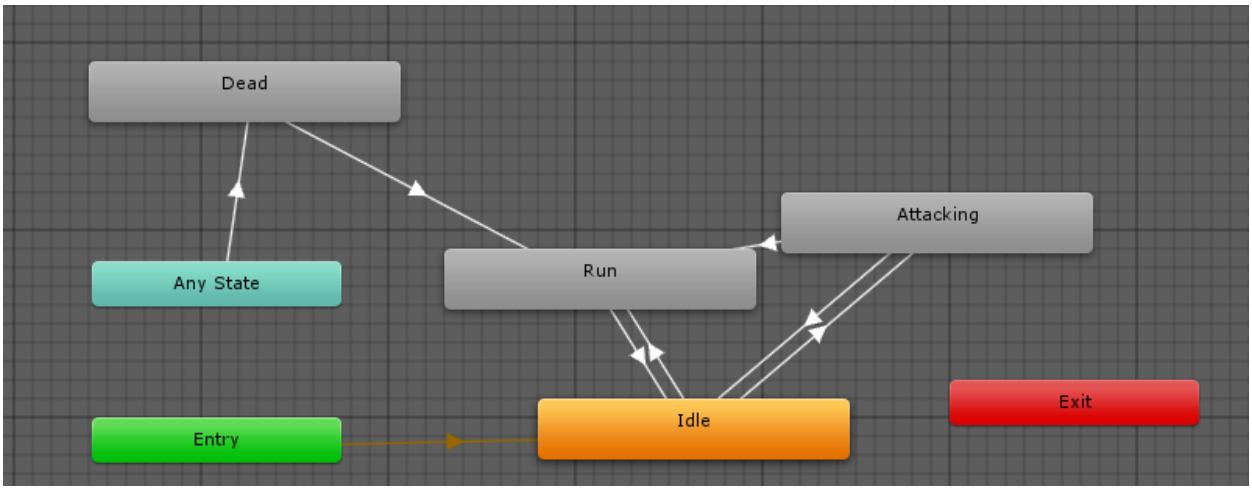


Figura 52: Animatorul jucătorului

Pentru a realiza animația personajului, s-a folosit componenta **Animator**. Acest mecanism de control se bazează 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 (7). De exemplu, pentru a porni animația de mișcare se va seta flagul **atDestination** pe fals.

Atacul unității se poate realiza doar dacă adversarii se află la o anumită distanță 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 observa legătura dintre cei patru pivoți ai jocului: **Idle**, **Run**, **Attack** și **Dead.**

4.4 Actori inteligenţi

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 desfășoară meciul, fiecare client va adăuga trei minioni activi și trei pasivi. Primii trei vor fi cei care vor detecta adversarii și rezultatele acestora 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 structurii *nexus* a adversarului.

Minionii activi sunt caracterizați de scripturile **CreepPlayer** și **AttackAI**. Acestea sunt folosite pentru a detecta și ataca adversarii în timpul deplasării acestora. Atunci când în raza de acțiune a fiecărui minion apare o entitate adversă, se va trimite către server id-ul acestuia dar și id-ul posibilei ținte. Serverul va trimite înapoi către clienți faptul că permite atacul. 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.

Figura 53: Funcția care caută inamic

```
private bool isReadyToScan()
{
    return (Time.time - lastAttackScan > scanRate && gameObject.tag != "EnemyMinions");
}

1 reference | astonished12, 33 days ago | 1 author, 8 changes
private void FindClosestEnemy()
{

    GameObject[] enemies = GameObject.FindGameObjectsWithTag("EnemyMinions");

    Vector3 position = transform.position;
    foreach (GameObject enemy in enemies)
    {
            float curDistance = GetDistanceBetweenPositions(enemy.transform.position, position);
            if (curDistance < minDistance)
            {
                if ((enemy.GetComponent<CreepAi>() && GetComponent<FollowerMinion>().mustStop == 0 && enemy.GetComponent<FollowerMinion>().mustStop == 0)
                  || ((GetComponent<CreepAi>() && GetComponent<FollowerMinion>().mustStop == 0 && enemy.GetComponent<CreepAi>()==null)))
                    {
                     GetComponent<FollowerMinion>().mustStop = 1;
                         if(enemy.GetComponent<CreepAi>())
                            enemy.GetComponent<FollowerMinion>().mustStop = 1;
                        GetComponent<NetworkCommunication>().SendMinionDataToFollow(GetComponent<NetworkEntity>().Id, enemy.GetComponent<NetworkEntity>().Id);
                        break;
                    }
                }
    }

}
```

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

```
1 reference | astonished12, 29 days ago | 1 author, 2 changes
private void OnFollowMinion(SocketIOEvent Obj)
{
    string socket_id = myJsonParser.ElementFromJsonToString(Obj.data["socket_id"].ToString())[1];
    string target_id = myJsonParser.ElementFromJsonToString(Obj.data["target_id"].ToString())[1];
    //remote
    var playerWhoDoRequest = spawner.OtherPlayersGameObjects[socket_id];
    //minion
    var target = spawner.minionsData[target_id];

    var followerOfPlaeryRequested = playerWhoDoRequest.GetComponent<Target>();
    followerOfPlaeryRequested.SetTargetTransform(target.transform);
}
```

Figura 54: Setarea posibilului adversar

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

```
1 reference | astonished12, 29 days ago | 1 author, 7 changes
private void OnMinionAttackMinion(SocketIOEvent Obj)
{
    string attacker_id_minion = myJsonParser.ElementFromJsonToString(Obj.data["id_attacker"].ToString())[1];
    string target_id_minion = myJsonParser.ElementFromJsonToString(Obj.data["target_id"].ToString())[1];

    if (spawner.minionsData.ContainsKey(attacker_id_minion) && spawner.minionsData.ContainsKey(target_id_minion))
    {
        var atacker_minion = spawner.minionsData[attacker_id_minion];
        var target_minion = spawner.minionsData[target_id_minion];
       if(target_minion.GetComponent<Alive>().isAlive && atacker_minion.GetComponent<Alive>().isAlive)
        {
            spawner.SpawnBullet(atacker_minion, atacker_minion.transform.position, target_minion.transform);
        }
    }
    else if(spawner.minionsData.ContainsKey(attacker_id_minion) && spawner.OtherPlayersGameObjects.ContainsKey(target_id_minion))
    {
        Debug.Log("Minionul " + attacker_id_minion + " urmareste jucatorul " + target_id_minion);

        var atacker_minion = spawner.minionsData[attacker_id_minion];
        var target_minion = spawner.OtherPlayersGameObjects[target_id_minion];
        if(target_minion.GetComponent<Alive>().isAlive)
            spawner.SpawnBullet(atacker_minion, atacker_minion.transform.position, target_minion.transform);
    }
    else if (spawner.minionsData.ContainsKey(attacker_id_minion) && spawner.towersData.ContainsKey(target_id_minion))
    {
        var atacker_minion = spawner.minionsData[attacker_id_minion];
        var tower_target = spawner.towersData[target_id_minion];
        if (tower_target.GetComponent<Alive>().isAlive)
            spawner.SpawnBullet(atacker_minion, atacker_minion.transform.position, tower_target.transform);
    }
}
```

Figura 55: Atacul unităților ajutătoare

Turnurile de control se comportă asemănator cu unitățile ajutătoare, cu excepția faptului că ele nu se deplasează. Există trei turnuri active și trei pasive. Detectarea inamicilor nu se face la fel ca la minioni.

Pentu a detecta posibilii adversari, se folosește componenta **BoxCollider** oferită de Unity. Astfel, când actorii adversari intra în zona turnului vor fi detectați prin suprapunerea cu o structură sferică.

Detectarea se va realiza la 360 de grade și la o anumită distanță maximă, 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 obținerea acestuia, se va trimite către server id-ul turnului și id-ul obiectului urmărit. (8)

```
void TargetEnemy()
{
    trackingObject = null;
    float hirange = float.MaxValue;
    Collider[] cols = Physics.OverlapSphere(transform.position, trackingRange);
    foreach (Collider col in cols)
    {
        GameObject target = col.gameObject;
        if (target != gameObject && target.CompareTag("EnemyMinions"))
                {
            Vector3 dir = target.transform.position - transform.position;
            float range = dir.magnitude;
            float angle = Vector3.Angle(dir, transform.forward);
            if (range <= hirange && angle <= fieldOfView)
            {
                hirange = range;
                trackingObject = target;
            }
        }
    }
    if (trackingObject)
    {
        GetComponent<NetworkCommunication>().SendMinionsOrPlayerIdToServerForTowerAttacking(GetComponent<NetworkEntity>().Id, trackingObject.GetComponent<NetworkEntity>().Id);
    }
}
```

Figura 56: Depistarea inamicilor de către turn

4.5 Atac magic

Fiecare jucător deține un numar de patru abilități ce vor fi folosite pentru a ataca adversarul. Aceste abilități sunt poziționate în bara de abilități ce este așezată în partea de jos a ecranului.

Atacul cu abilități magice se poate realiza de la distanță sau de la apropiere. Abilitățile pot fi folosite de la orice nivel doar că, pentru utlizarea lor sunt necesare punctele de *mana*. Odată folosită o abilitate, pentru a putea reutiliza efectul acesteia trebuie așteptat timpul de reîncărcare. Fiecărei abilități îi corespunde o anumită tastă pentru a putea declanșa efectul.

Abilitățile **Q** și **R** sunt abilități ce pot produce daune la distanță/apropiere.

Abilitatea **W** are efect asupra utilizatorului oferindu-i acestuia un anumit număr de puncte de viață. Abilitatea **E** este utilzată în lupta de apropiere. Ea provocă daune doar dacă adversarul este în apropiere.

```
public class SkillSlot
{
    public SpecialAttack skill;
    public Rect positionRect;
    public KeyCode key;

    public bool coolDownActive=false;
    public float coolDonwTime;
}
```

Figura 57: Structura SkillSlot

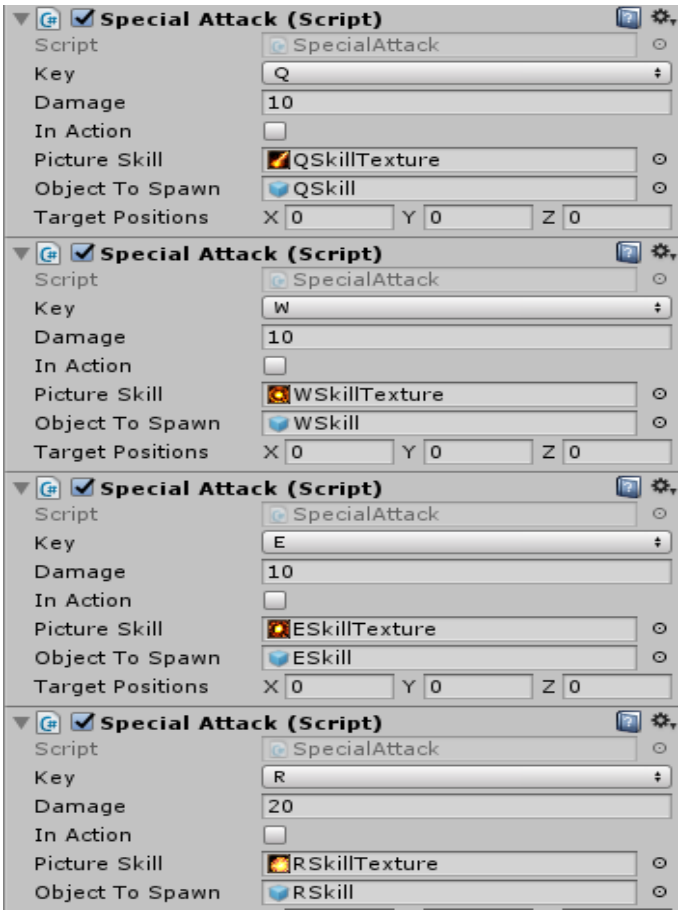


Figura 58: Componentele SpecialAttack ale campionului

Fiecărei abilități îi este rezervat un loc în bara specifică atacului magic. Pentru a menține idea de joc sincronizat, abilitățile sunt declanșate dupa ce serverul va permite acest lucru.

Abilitatea Q

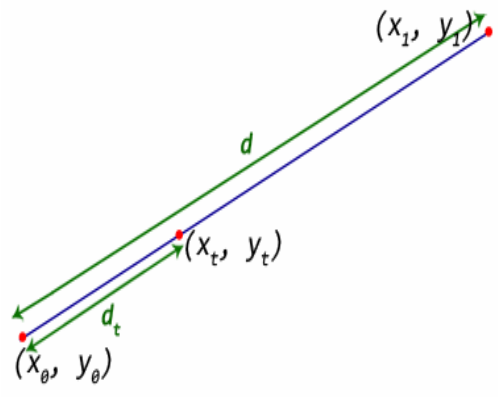
Aceasta abilitate se poate folosi pentru a lovi adversarul pe poziția acestuia sau pe o poziție viitoare, la alegerea clientului. Abilitatea are o rază de atac stabilită. În cazul în

care, punctul de interacțiune al hărții se află la o distanță mai mare decat limita stabilită, se calculează punctul aflat la distanța maximă posibilă față de poziția curentă a unității campion.

Pentru a determina punctul aflat la distanța maximă vom folosi vectori:

$$\mathbf{v} = (x_1, y_1) - (x_0, y_0). \text{ Normalizăm vectorul către } \mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|}.$$

Punctul aflat pe linie la distanța *d* față de punctul (x0,y0) se va calcula folosind **(x0,y0) + du(x0,y0)** în direcția punctului (x1,y1) sau **(x0,y0) - du(x0,y0)** în direcția opusă. (9)



Normalizarea vectorului se face astfel:

$$\frac{v}{\|v\|} = \left(\frac{v1}{\sqrt{v1^2 + v2^2}}, \frac{v2}{\sqrt{v1^2 + v2^2}} \right), \text{ unde } \|v\|$$

este definită ca fiind lungimea vectorului v.

Figura 59: Determinarea punctului (x1, y1)

Astfel abilitatea este declanșată la coordonatele obținute **(x1, y1).**

```
if (GetDistanceBetweenPositions(transform.position, targetPositions) > distance)
{
    Vector2 start = new Vector2(transform.position.x, transform.position.z);
    Vector2 end = new Vector2(targetPositions.x, targetPositions.z);
    targetPositions = MakeQSpawnPointTarget(distance, transform.position, GetNormalizeStartToEnd(start, end));
}
```

Figura 60: Setarea coordonatelor de impact ale abilității

4.6 Autoritatea serverului

Dezvoltarea oricărui tip de joc este ea însăși provocatoare. Pentru a dezvolta un joc bun trebuie respectat un anumit plan. În acest plan, trebuie luate în considerare elementele fizice, mișcările acestora, dar și posibilele contacte ce s-ar putea realiza între ele. Toate aceste lucruri presupun deja o muncă grea. Totuși, pentru realizarea

unui joc multiplayer se adaugă un nou set de probleme ce vor face ca dezvoltarea să fie mai complexă. O problemă de luat în considerare este menținerea comunicării în timp real cu serverul astfel încat să nu existe pe cât posibil întarziere la rețea.

O altă problemă importantă pentru jocurile multiplayer ce trebuie tratată este încercarea jucătorilor de a trișa. Atunci când se dezvoltă un joc în manieră single-player, dezvoltatorul nu se interesează prea mult dacă jucătorii vor trișa. Acțiunile acestora nu afectează pe nimeni altcineva decât pe ei. Experiența jocului nu va mai fi așa cum a fost plănuită de la început.

Jocurile multiplayer sunt diferite la acest capitol. În orice joc competitiv, un jucător care trișează face ca experiența jocului să fie una proastă nu doar pentru el, dar și pentru ceilalați jucători. Atunci când se dezvoltă un joc, se dorește evitarea acestei probleme. Dacă un utilizator va trișa, este posibil ca acest lucru ii va face pe ceilalți să părăsească aplicația.

Pentru a preveni trișarea, se poate rula motorul jocului pe server și tot ceea ce se întâmplă sub controlul său. Astfel, clienții vor deveni doar niște simpli spectatori ai jocului. Cu alte cuvinte, clientul trimite comenzi la server, serverul rulează jocul și va trimite înapoi clienților rezultatele.

Această soluție se numește folosirea server-ului autoritar, pentru că server-ul este singura autoritate cu privire la ceea ce se întâmplă în joc. Dar, acest lucru poate fi destul de ineficient dacă vorbim de un numar mare de coliziuni, jucători și evenimente de tratat. (10)

În cadrul jocul BattleOfTitans, acțiunea se desfășoară pe partea clientului și nu pe partea serverului. Deși algoritmii sunt implementați pe parte de client, acest lucru nu impiedică validarea datelor trimise de la clienți către server. Aceste date sunt memorate, actualizate și verificate la fiecare eveniment, pentru a oferi un minim de autoritate serverului.

Acestă metoda prespune faptul că clienții nu pot fi de încredere și va trebui verificată fiecare acțiune a acestora, pe partea serverului. Atunci când se verifică datele, trebuie luate în considerare momentele transmiterii și recepționării, pentru a calcula durata dintre evenimente.

În continuare voi prezenta cum se realizează verificarea mișcării unității campion la fiecare eveniment. Fiecărui jucător ii corespunde un obiect de tipul **ControllerPlayer**.

```
var   ControllerPlayer = function (id, name, _x,_y,_z,isOwner) {
        this.id = id;
        this.name = name;
        this.destination = {
            x : 0,
            y : 0,
            z : 0
        }
        this.lastPosition = {
            x : _x,
            y : _y,
            z : _z
        }

        this.totalDamage = 0;
        this.win = false;
        this.speed = 5;
        //TO POSITION ON MOVE UPDATE
        this.isOwner = isOwner;
        this.characterNumber = -99;
        this.lastMoveTime = 0;
    };

ControllerPlayer.prototype.updateDestination = function(data){
    this.destination.x = data["x"];
    this.destination.y = data["y"];
    this.destination.z = data["z"];
};
ControllerPlayer.prototype.updateLastPosition = function(data){
    this.lastPosition.x = data["x"];
    this.lastPosition.y = data["y"];
    this.lastPosition.z = data["z"];

module.exports = ControllerPlayer;
```

Figura 61: Clasa ControllerPlayer pe server

Atunci când un client dorește mutarea unității campion pe hartă, se va trimite către server poziția curentă și poziția destinație. Pe partea serverului, se va memora timpul dintre două mișcări consecutive.

Se va calcula distanța în linie dreaptă, folosindu-se timpul obținut mai devreme și campul **speed**. Folosindu-se poziția *curentă* și **lastPosition** se va calcula distanța solicitată de către client. În cazul în care distanța solicitată de catre client este mai mare decat distanța calculată în linie dreaptă este depistat trișorul. Altfel se vor actualiza noile poziții pe server și se va trimite semnalul de mișcare catre clienții aflați în cameră.

```
var epsilon = 0.25;
var onMoveClient = function(data){

    mapingSocketRoom[this.id].PLAYERS[this.id].updateDestination(data["destination"]);

    var elapsedTime = Date.now() - mapingSocketRoom[this.id].PLAYERS[this.id].lastMoveTime;
    var posibleTravelLimit =  mapingSocketRoom[this.id].PLAYERS[this.id].speed * elapsedTime/1000;
    var requstedDIstanceTraveled = lineDistance(mapingSocketRoom[this.id].PLAYERS[this.id].lastPosition,data.current);

    mapingSocketRoom[this.id].PLAYERS[this.id].lastMoveTime = Date.now();
    mapingSocketRoom[this.id].PLAYERS[this.id].updateLastPosition(data["current"]);

    if(requstedDIstanceTraveled>posibleTravelLimit+epsilon){
       io.to(mapingSocketRoom[this.id].name).emit("hacking",{socket_id:this.id});
       //console.log("Hacking");
    }
    else
    {
    this.broadcast.to(mapingSocketRoom[this.id].name).emit("playerMove",{
        socket_id:this.id,
        x : data["destination"]["x"],
        y : data["destination"]["y"],
        z : data["destination"]["z"],
    });
}
};
```

Figura 62: Funcția onMoveClient pe server

5 Manual de utilizare

5.1 Login

Prima interacțiune pe care utilizatorul o are cu aplicația este meniul de autentificare.

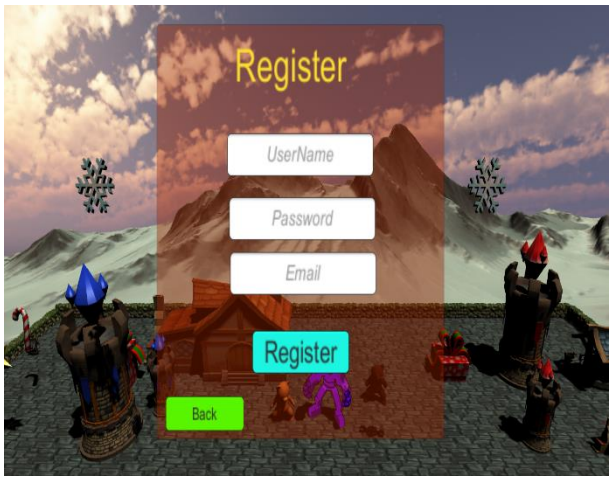
Pentru a putea avea acces în aplicație, utilizatorul se va autentifica la server pe baza unui cont și al unei parole. Utilizarea aplicației fară autentificare nu este posibilă. Dupa primirea confirmării de la server, utilizatorul va fi redirecționat către meniul jocului.

Figura 63 și 64: Meniul de autentificare

5.2 Register

Pentru a putea ajunge la acest ecran, utilizatorul se va folosi de butonul verde aflat pe ecranul de autentificare. Aici există 3 câmpuri necesare pentru a se realiza operația de înregistrare: *username*, *password*, *email*. Câmpul *username* va fi utilizat peste tot în aplicație dar și în joc.

Există de asemenea posibilitatea de revenire către meniul de autentificare. După înregistrare utilizatorul va fi redirecționat la fel către același meniu.



Figurile 65 și 66: Meniul de înregistrare

5.3 Main Menu

Acesta este ecranul principal al aplicației .Utilizatorul va putea să acceseze istoricul meciurilor desfășurate, sa pornească o conversație cu unul dintre prietenii săi, să creeze o nouă cameră pentru a porni un meci sau să se alăture uneia deja create.

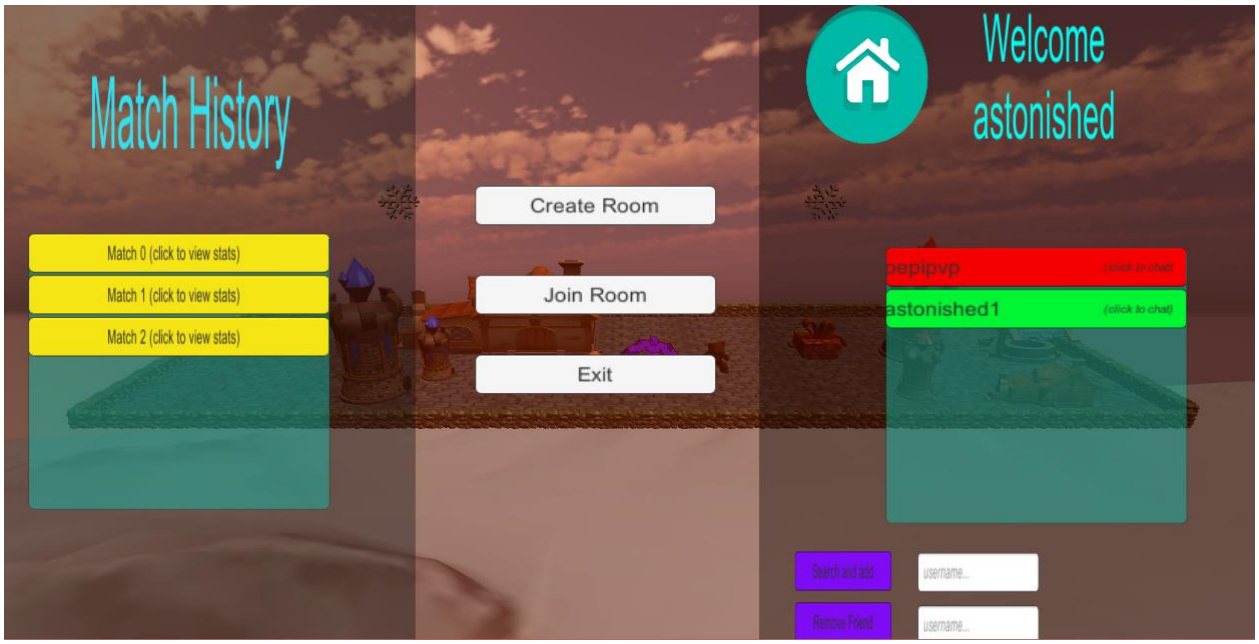
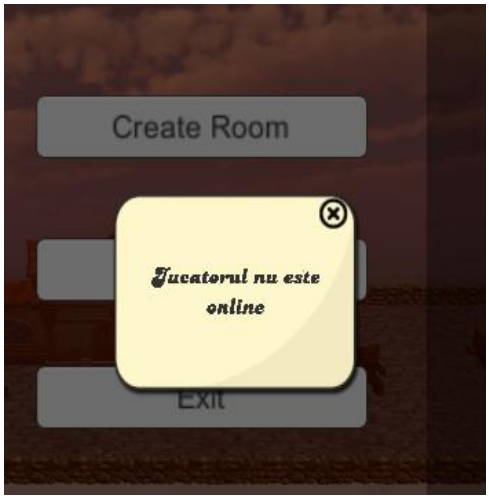


Figura 67: Meniul principal al aplicației

În partea stânga, este lista meciurilor desfășurate. În cazul în care utilizatorul dorește să acceseze un meci, se va deschide un dialog cu statisticile acestuia. În partea dreaptă, este lista prietenilor activi și inactivi. Cei activi sunt reprezentați intr-un chenar verde iar cei inactivi intr-un chenar roșu. Tot aici, există posibilitatea de a adauga sau elimina un prieten din listă.

Pentru a porni o conversație, se va acționa mouse-ul asupra unui prieten activ. Conversația nu poate porni dacă prietenul este inactiv.



Figurile 68 și 69: Pornire chat global

Crearea și alăturarea la camere se face acesând butoanele principale ale ecranului. Cel care creează camera va aștepta ca un alt client să i se alăture. Cel care dorește să se alăture, va putea alege din lista de camere existente pe server.

Figura 70 și 71: NewRoom/JoinRoom

5.4 Character Selection

După ce camera va conține 2 jucători, serverul va trimite către clienți semnalul de pornire al jocului. Clienții vor fi redirecționați către meniul de alegere al campionilor. Cei doi vor alege campionii cu care vor intra în joc.

Figura 70 și 71: Selectarea campionilor pe fiecare client

5.5 Game

Figura 72 și 73: Imagini din joc client1 și client2

Figura 74: Imagine din joc ce prezintă unitățile ajutătoare și un turn de protecție

Ecranul Game reprezintă ecranul unde se va desfășura jocul propiu-zis. În acest moment, fiecare jucător se poate deplasa pe hartă și va putea interacționa cu celalalt adversar sau construcții adverse. Câștigător va fi considerat jucătorul care distruge primul contrucția principală a adversarului. Pentru deplasare, se va folosi click-ul 2 al mouse-ului. Abilitățile principale sunt declanșate prin apasarea tastelor *Q+click1*, *W, E, R.*

Figurile 75 și 76 : Atacul turnului cu steluța și Q skill atac – perspectiva celor 2 clienți

Figura 77 : Atac magic cu abilitatea E

Figura 78 : Abilitatea de vindecare W

Figura 79 : Atac magic cu abilitatea R

Chatul din joc se pornește apăsând tasta *ENTER*, iar pentru a trimite mesajul se mai execută odată.

5.6 End Game

Acest ecran apare la final când unul dintre clienți distruge construcția finală. Există un ecran ce sugerează victoria dar și un ecran ce sugerează înfrangerea. Pentru a ajunge la meniul principal, se acționează buton din centru.

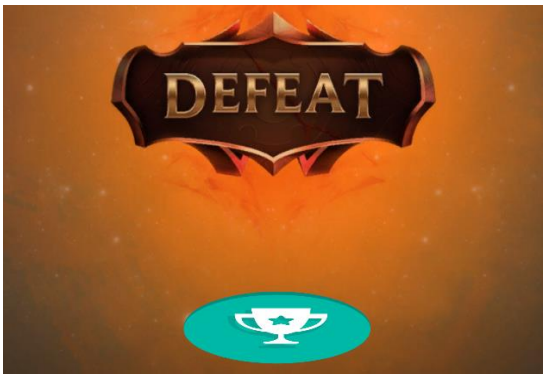


Figura 80 și 81: Victorie sau înfrângere

6 Concluzii

Scopul acestei aplicații este acela de a oferi utilizatorului o perspectivă mai amuzantă asupra jocurilor din categoria MOBA. Personajele, unitățile ajutătoare, mișcările acestora accentuează acestă idee. Jocul poate fi jucat intr-o perioadă scurtă de timp, chiar dacă aplicația a fost dezvoltată doar pe *Windows.*

Pentru realizarea jocului au fost folosite câteva elemente principale, dintre care: aplicarea unor algoritmi de inteligență artificială pentru mecanismele personajelor, dobândirea noțiunilor despre arhitectura unui joc multiplayer și sincronizare a datelor, evitarea, pe cât posibil, a încercării de a trișa a jucătorilor. Toate aceste lucruri s-au combinat cu utilizarea animațiilor grafice oferite de motorul de dezolvoltare a jocurilor Unity.

De asemenea, pe partea serverului au fost necesare cunoștințe despre utilizarea bibliotecii *Socket.IO* oferită de *Node.Js, c*u aceasta realizand comunicarea în timp real dar și serviciul camerelor.

Extensii

Deși jocul în sine nu oferă o grafică uluitoare, aceasta poate fi îmbunătățită. Totodată, pe partea de multiplayer se mai pot adăuga alte personaje principale, dar și unele creaturi care în timpul meciurilor pot oferi anumite bonusuri prin interacțiunea cu acestea. Se poate contribui și la sistemul de dezvoltare al abilităților și al nivelurilor.

Pe viitor, jocul ar putea fi îmbunătățit prin dezvoltarea unui stil de joc singleplayer. Se poate implementa un sistem de inteligență artificială cu anumite niveluri de dificultate ce va folosi un algoritm nedeterminist care se va adapta la cerințele selectate de utilizator.

7 Bibliografie

1. The history of MOBAs: From mod to sensation. [Online] https://venturebeat.com/2014/09/01/the-history-of-mobas-from-mod-to-sensation/.

2. Pagina oficială Node.js. [Online] https://nodejs.org/en/.

3. A* Pathfinding Algorithm. [Online] http://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html.

4. A*'s Use of the Heuristic. [Online] http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#a-stars-use-of-the-heuristic.

5. [Online] https://github.com/SebLague/Pathfinding.

6. Heaps. [Online] http://faculty.cs.niu.edu/~freedman/340/340notes/340heap.htm.

7. Animator Unity. [Online] https://unity3d.com/learn/tutorials/topics/animation/animator-controller.

8. [Online] https://unity3d.com/learn/tutorials/topics/physics.

9. [Online] https://math.stackexchange.com/questions/175896/finding-a-point-along-a-line-a-certain-distance-away-from-another-point.

10. [Online] http://www.gabrielgambetta.com/fpm1.html.