MINISTERUL EDUCAȚIEI ŞI CERCETĂRII ŞTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

FLIB – Joc multiplayer online de acțiune

LUCRARE DE LICENŢĂ

Absolvent: **Sergiu Fălcușan**

Coordonator ştiinţific: **Ș. l. ing. Cosmina IVAN**

2016

MINISTERUL EDUCAȚIEI ŞI CERCETĂRII ŞTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

DECAN,

Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,

Prof. dr. ing. Rodica POTOLEA

Absolvent: **Sergiu Fălcușan**

FLIB – Joc multiplayer de acțiune

1. **Enunţul temei:** *Proiectul își propune realizarea unui joc multiplayer de acțiune în care un număr mare de jucători să lupte între ei pentru a ajunge primii în clasament.*

2. **Conţinutul lucrării:** *Cuprins, Introducere - Contextul Proiectului, Obiectivele proiectului, Studiu bibliografic, Analiza şi fundamentare teoretică, Proiectare de detaliu şi implementare, Testare şi validare, Manual de instalare şi utilizare, Concluzii*

3. **Consultanți**:

4. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

5. **Data emiterii temei: 1 Martie 2016**

6. **Data predării: 17 Februarie 2017**

Absolvent: Sergiu Fălcușan

Coordonator ştiinţific: Ș. l. ing. Cosmina IVAN

Cuprins

Capitolul 1. Introducere – Contextul Proiectului

În acest capitol se va face introducerea aplicației Flib, prin prezentarea contextului datorită căruia s-a început implementarea ideii precum si motivația din spatele acesteia. Se va prezenta în câteva rânduri și conținutul lucrării.

1.1. Contextul proiectului

Oamenii caută divertismentul, caută metode de relaxare și ocupații cu care să își acopere timpul când sunt liberi și să se distreze. Pentru majoritatea oamenilor, jocurile sunt una din ocupațiile care le permite să se relaxeze și să își dezvolte imaginația, iar in unele cazuri pot ajuta la vindecarea depresiilor.

Primele jocuri pe calculator au apărut în anii 1970. Datorită ușurinței utilizării calculatoarelor, ele au căpătat o popularitate imensă ceea ce a dus la perioada de aproximativ un deceniu numită și epoca de aur a jocurilor Arcade. Evoluția masivă a calculatoarelor în ultimele decenii și apariția Internetului au dus în același timp și la evoluția jocurilor. Așadar în prezent pe lângă faptul că jocurile au o grafică vizuală aproape indistinctivă de realitate, jucătorii pot interacționa în timp real unii cu ceilalți.

1.2. Motivația

Flib vrea să ofere fiecărui jucător o metodă de relaxare în timpul liber cât și totodată plăcerea de a concura împotriva altor persoane pentru a deveni cel mai bun. Aplicația permite utilizatorului de a-și demonstra îndemânările și abilitățile strategice. Neavând un sfârșit predefinit, o rundă de joc depinde doar de abilitățile jucătorului și de timpul pe care îl decide că vrea să îl investească pentru divertisment. Fie că utilizatorul este un jucător profesionist sau este un jucător de ocazie, jocul nu favorizează pe nimeni, neoferind beneficii celor care joacă de o perioadă mai mare de timp. Astfel toată lumea este la fel de motivată să se joace.

1.3. Conținutul lucrării

În continuare se va prezenta conținutul acestei lucrări. Acest document conține următoarea structură:

- În Capitolul 1 s-a prezentat contextul și motivația din spatele dezvoltării acestui proiect care stau în strânsă legătură.
- Capitolul 2 prezintă scopul acestei lucrări. Astfel se vor prezenta obiectivele, problemele și soluțiile care pot fi luate în considerare în scopul obținerii performanțelor dorite și în ultima parte va fi prezentată poziționarea produsului în cadrul din care face parte.
- Capitolul 3 descrie studiul bibliografic efectuat dinaintea începerii implementării aplicației. Se vor prezenta istoria și bazele dezvoltării jocurilor, arhitectura necesară și un studiu privind sistemele similare, comparându-le cu acest produs dezvoltat în lucrare.
- Capitolul 4 prezintă o analiză și o fundamentare teoretică în urma studiilor

aprofundate asupra tehnologiilor folosite cu argumente pentru folosirea acestora. Tot în acest capitol au fost prezentate și cerințele funcționale și nonfuncționale, precum și cazurile de utilizare.

- Capitolul 5 descrie pașii parcurși ai etapelor de proiectare și dezvoltare ai aplicației atât pentru partea de backend care trebuie să ruleze pe un server cât și pentru interfața grafică care va rula în browserul utilizatorului.

- Capitolul 6 prezintă tehnicile de testare și validare folosite în cadrul dezvoltării aplicației și un rezumat al răspunsurilor oferite de o parte din utilizatorii care au testat aplicația.

- Capitolul 7 conține informații despre cerințele sistemului și pașii care trebuie parcurși pentru instalarea și rularea aplicației.

- Capitolul 8 reprezintă ultimul subiect abordat care oferă o concluzie asupra obiectivelor care au fost realizate și a posibilităților de dezvoltare ulterioară.

Capitolul 2. Obiectivele proiectului

În acest capitol se vor prezenta obiectivele propuse spre realizare prin această temă. Sistemul are scopul de a oferi utilizatorilor un joc care să prezinte o metodă de relaxare și divertisment.

2.1. Obiectivul principal

Obiectivul principal al acestui proiect este de a proiecta, defini și construi o aplicație client-server care să reprezinte un joc de acțiune multiplayer-online în cadrul unui browser web care să ofere o metodă alternativă de divertisment persoanelor care doresc să participe într-o competiție continuă alături de alți jucători considerați oponenți.

Un astfel de sistem dezvoltat pe tehnologii web trebuie să fie capabil de a face actualizări ale interfeței utilizator de aproximativ 30-60 de ori pe secundă și în același timp de a comunica în timp real cu un server, care calculează stările tuturor jucătorilor. De asemenea, fiind joc online multiplayer, este nevoie ca sistemul să se scaleze automat în funcție de câți jucători se vor conecta. Securitatea este foarte importantă, deoarece utilizatorul nu trebuie lăsat să păcălească sistemul prin diferite metode de hacking, astfel toate mișcările trebuie verificate pe server la fiecare pas.

2.2. Specificarea problemei

În paragraful următor sunt descrise problemele și soluțiile, însoțite de componentele afectate, care pot fi luate în considerare în scopul de a obține performanțele așteptate.

Prima problemă este comunicarea dintre client și server. Deoarece jocul trebuie să ruleze la o frecvență de cadre foarte mare, o implementare clasică nu este suficientă. Frecvența cadrelor influețează foarte mult decursul jocului și opinia celor care se joacă. De asemenea ținând cont că pe backend se va executa o mare parte din logica jocului, toți algoritmii folosiți vor trebui optimizați foarte bine astfel încât să se piardă cât mai puțin timp în servirea actualizărilor spre client.

Arhitectura la baza sistemului va fi una client-server între care comunicarea se va face printr-un websocket care va ramane deschis pe tot parcursul jocului. Această tehnică ne scutește de metoda alternativă de a folosi request-uri clasice (GET/POST) care ar consuma mai multe resurse nenecesare în acest caz, iar durata unui request fiind semnificativ mai mare.

O altă problemă care poate apărea este aceea că un număr mare de utilizatori ar putea aglomera foarte tare procesorul serverului, aceasta ar duce totodată la servirea actualizărilor către utilizatori cu întârziere.

O soluție a fost împărțirea jucătorilor în mai multe camere de joc și cu ajutorul unei componente care se ocupă de echilibrarea sarcinilor (load balancer), se pot împărți camerele de joc pe mai multe servere.

Ultima problemă care va fi evidențiată este legată de securitatea aplicației. Ca și în cadrul oricărei aplicații web, datele din browser și care sunt trimise către server sau alți

jucători pot fi modificate foarte ușor.

De aceea una din soluțiile plauzibile este ca logica să fie ținută pe server și nu local în browser ca și în marea majoritate a jocurilor web. Astfel clientul va trimite update-uri de la tastatură și mouse către server, iar serverul va calcula următoarea stare care va fi trimisă ulterior jucătorilor și afișată.

2.2. Poziționarea produsului

Produsul face parte din gama jocurilor de acțiune multiplayer online dezvoltat în browser web ceea ce oferă posibilitatea de a extinde aplicația pe toate deviceurile, indiferent de sistemul de operare, care au o aplicație de browser și suportă HTML5, Javascript ES6 și WebSockets. În tabelul 2.1 este detaliat target-ul aplicației, scopul și diferențele față de alte soluții.

Poziționarea produsului	
Destinat	uz general, jucători profesioniști
Ce doresc	a se relaxa printr-o metodă de divertisment și a competiționa cu alți jucători
Cu scopul de a	demonstra abilitățile de agilitate și dexteritate
Spre deosebire de alte soluții	Flib, oferă o arhitectură care suportă un număr ridicat de jucători cu algoritmi optimizați pentru cerintele problemei. Astfel comunicarea dintre server si client se face la o viteza mare care permite ca jocul să pară real-time.
Acest produs	oferă o arhitectură client-server, care scade riscul ca hackerii să creeze tooluri care să ii ajute și să le ofere avantaje asupra altor jucători

Tabel 2.1 – Poziționarea produsului

Capitolul 3. Studiu Bibliografic

În continuare vor fi detaliate conceptele și toolurile care au fost studiate în scopul dezvoltării jocului Flib. La început se va descrie background-ul și istoria proiectării jocurilor, mai apoi va fi prezentat protocolul websocket care este foarte popular și folosit în jocurile online.

3.1. Istoria și bazele dezvoltării și proiectării unui joc

În prima fază a devoltării industriei de jocuri au apărut jocurile Arcade [1]. Datorită evoluției în tehnologiile multimedia, jocurile video au devenit tot mai predominante printre jucători din toată lumea. O dată cu apariția jocurilor online în anii 1970, ele au început să domine piața online cu o penetrare tot mai adâncă în tehnologii și în viețile oamenilor. Jocurile domină de asemenea piața online, marea majoritate a cotelor de pe piață o reprezintă jocurile online.

Cererea pentru jocuri nu este doar pentru posesorii de pc-uri, ci de asemena și în aria de mobile (telefoane și tablete). Figura de mai jos (Figura 3.1) aparține unui studiu [2] care prezintă faptul că timpul petrecut de oameni în jocuri este predominant față de alte aplicații. Conform acestui studiu, 32% din timpul petrecut pe aceste deviceuri este folosit în scop de gaming.

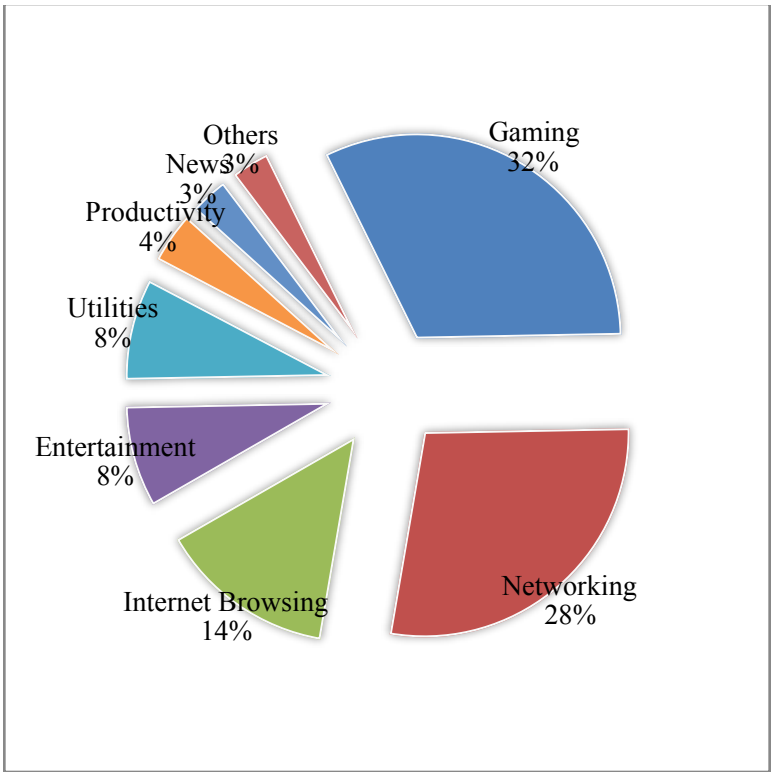


Figura 3.1 – Statistică a timpului petrecut la calculator și alte deviceuri

Jocurile au avut o influență majoră în evoluția sistemelor performante și a tehnologiilor, iar în momentul de față cererea este tot mai mare, iar producătorii încearcă să țină pasul dezvoltând diferite componente hardware tot mai puternice. De asemenea dezvoltatorii de jocuri sunt într-o continuă cerere, dar datorită pieței, se pune mare accent și pe crearea de framework-uri și tooluri, astfel ușurând procedura de dezvoltare a jocurilor.

Proiectarea unui joc este arta de a crea caractere și obiecte pentru lumea digitală. Dezvoltarea sa începe cu o idee care poate reprezenta diferite întâmplări și evenimente. Următorul pas este stabilirea designului. Designul jocurilor în general este creat cu ajutorul a foarte multe ecuații matematice și fizice, designerii construiect și transformă lumile imaginare în spațiu digital. Datorită toolurilor care există deja, designul jocurilor este foarte ușor de implementat.

Jocurile sunt proiectate cu ajutorul ecuațiilor și teoriilor matematice [3] care în general folosesc reprezentări matriceale sau arbori pentru rezolvarea anumitor probleme specifice și de optimizare. De asemenea jocurile au nevoie de câteva elemente de bază. Acestea sunt jucătorii, informațiile și acțiunile fiecărui jucător la fiecare punct stabilit și reacțiile la fiecare eveniment. Bazat pe aceste elemente se stabilesc seturi de strategii pentru echilibrare jocului intr-un mod în care nici un player să nu fie avantajat. Datorită acestui set de strategii pentru echilibrare, un joc poate fi considerat într-o stare stabilă care duce la un feedback pozitiv și la un eventual profit.

3.2. Arhitectura și componentele unui joc multiplayer

Odată cu avansarea tehnologiilor și accesul la Internet, s-a putut observa creșterea masivă a interesului jucătorilor față de jocurile online. Un joc online implică conectarea mai multor jucători din diferite părți ale lumii la un server oferind posibilitatea jucătorilor să interacționeze unul cu celălalt de la distanță. Acestă interacțiune între jucători a devenit o obsesie pentru companiile mari de jocuri, devenind un trend al industriei. Conexiunea și comunicarea între playeri se poate face cu ajutorul unor seturi de servere intermediare fie remote hostate de diferite companii cu ajutorul a diferite planuri, în funcție de cerințele clientului, fie implementate local folosind tehnologii peer-to-peer.

3.3. Comunicare în timp real prin websockets

Conform definiției [4], comunicarea în timp real (RTC) este un mod de telecomunicații în care utilizatorii pot schimba informații instant sau cu un efect întârziat neglijabil. În continuare se vor prezenta tehnologii RTC care au fost folosite în proiect: protocolul WebSocket, un protocol full-duplex care poate transmite date bidirecțional simultan pe același circuit, iar mai apoi va fi prezentată și librăria Javascript Socket.IO care conține o implementare a protocolului WebSocket.

3.3.1. Protocolul WebSocket

WebSocket [4] este un protocol care oferă canale de comunicații bidirecționale simultan (full-duplex) printr-o singură conexiune TCP. Este suportat de majoritatea

browserelor moderne incluzând Google Chrome, Firefox, Safari, Opera și Internet Explorer, iar pentru a funcționa se cere ca și serverul să accepte conexiuni de tip WebSocket.

Protocolul a fost proiectat pentru a putea fi folosit de către browserele web pentru comunicarea cu serverul sau alți clienți fără a fi nevoie de a deschide mai multe conexiuni. De asemenea spre deosebire de alte protocoale, acesta poate să țină canalul dintre server și client deschis până una dintre părți închide conexiunea. În timpul în care canalul este într-o stare deschisă, se poate comunica real-time între cele doua părți, astfel transferându-se fluxuri de date atât de la server la client cât și de la client la server simultan.

Pentru a face un handshake, clientul tebuie creeze un request de tip WebSocket, iar serverul va returna un raspuns de tip WebSocket handshake. Mai jos este prezentat un exemplu de request (Exemplul 3.1) de la un client care dorește conectarea prin websocket și un răspuns (Exemplul 3.2) din partea serverului.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Figura 3.2 – Client-Request WebSocket

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

Figura 3.3 – Server-Response Websocket

3.3.2. Socket.io

Socket.io [5] este un framework scris în Javascript pentru aplicațiile Node.js și web care vor să beneficieze de comunicarea în timp real. Acest framework abstractizează diferite protocoale de transport al datelor și oferă un API pentru a le accesa. Pentru o mai

bună înțelegere este important să se sublinieze faptul că acest framework are implementări diferite atât pentru partea de backend în Node.js cât și pentru partea de frontend din browser scris în Javascript.

Principala metodă de comunicare este prin WebSockets, dar în cazul în care comunicarea nu este suportată, sau din alte motive nu este posibilă, librăria știe să se retragă la o altă metodă de comunicare, fără a fi necesară o implementare extra din partea developerului. Metodele suportate sunt WebSockets, Adobe Flash Socket, AJAX long polling, AJAX multipart streaming, Forever Iframe and JSONP Polling. Astfel Socket.io permite folosirea de comunicații în timp real atât în browserele mai noi folosindu-se de puterea maximă din protocoalele mai rapide (WebSocket) cât și browserele mai vechi, alegând metoda cea mai rapidă în funcție de posibilități.

Implementarea unui sistem realtime folosind această librărie necesită cel puțin un server care să aștepte și să asculte pentru eventuale conexiuni de la clienți. În momentul în care un utilizator este conectat la server, se crează o conexiune între ei care rămâne deschisă până unul dintre ei anunță închiderea conexiunii. În tot acest timp, pe conexiunea deschisă serverul și clientul vor comunica trimițându-și diferite mesaje care pot fi intepretate în mod partiular.

Există două metode mai importante care sunt folosite in Socket.io: on și emit. Pe server, metoda on va asculta pentru mesaje din partea clientului, iar emit va transmite mesaje către client. O implementare asemănătoare este făcută și pentru partea de fronted pentru a comunica cu serverul. În exemplul 3.3 va fi prezentată o posibilă implementare prin care se va inițializa o conexiune între cele două părți, urmată de un mesaj din partea serverului care cere clientului să se autentifice, iar mai apoi o metodă care așteaptă un mesaj de autentificare din partea clientului.

```
io.on('connect', function (socket) {
    socket.emit('astept autentificare');

    socket.on('autentificare', function (data) {
        socket.emit('welcome');

        //....
    });

    // ....
});
```

Figura 3.4 - Creare conexiune și autentificare utilizator pe server

O altă funcționalitate foarte importantă care este oferită de socket.io sunt camerele. Camerele reprezintă o grupare a clienților ceea ce oferă posibilitatea de a emite anumite evenimente unui grup cu un număr mai restrâns de utilizatori, respectiv a utilizatorilor care sunt prezenți în camera respectivă. Într-un joc online, camerele ajută la separarea canalelor la care un utilizator se poate conecta, astfel putându-se scala aplicația mai ușor în funcție de numărul de utilizatori prezenți.

3.4. Sisteme similare

3.4.1. Agar

Un joc asemănător cu cel implementat în această lucare este Agar [6], care într-un timp record a reușit să fie nominalizat pentru titlul de ”jocul anului”[7], fiind jucat de câteva mii de persoane zilnic.

Agar este un joc multiplayer de acțiune în care jucătorii controlează o celulă într-un vas Petri (folosit în chimie pentru studiul celulelor). Scopul jocului este de a crește propria celulă în greutate, mâncând alte celule (alți jucători) de dimensiuni mai mici fără a fi înghițit de alte celule mai mari. Jocul oferă posibilitatea de a separa celula proprie în mai multe celule mai mici, oferind anumite beneficii utilizatorului, dar fiind șanse mai mari de a fi mâncate de alții.

Datorită feedback-ului primit de la jucători, acesta a fost îmbunătățit și optimizat foarte mult, iar în momentul de față, jocul este disponibil atât în browser cât și pe telefoanele mobile. Criticii spun că jocul este foarte repetitiv și poate deveni plictisitor la un moment dat, dar în contrast cu acești critici, multe persoane apreciaza simiplitatea și competiția.

Simplitatea și competiția este ceea ce se dorește a fi văzut în ochii oamenilor și în Flib. Ținând cont de criticile jocului Agar, în Flib, se dorește posibilitatea de personalizare a jucătorului în funcție de resursele consumate.

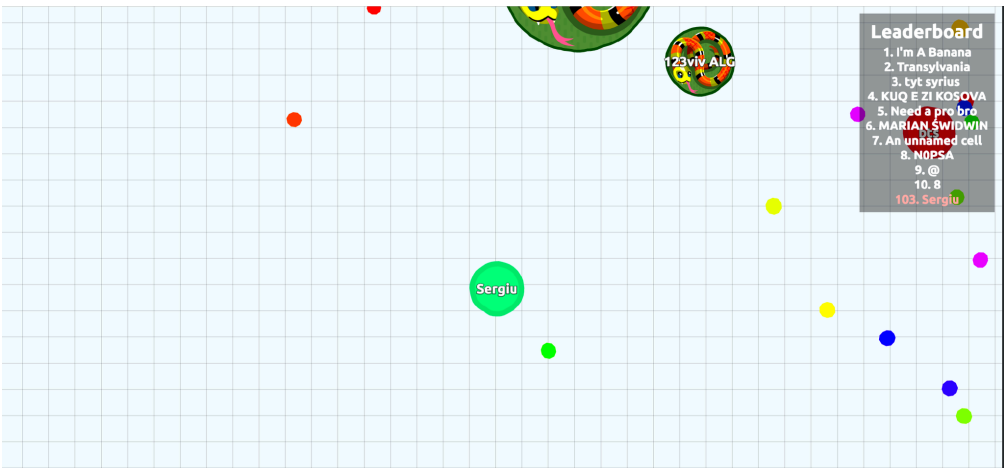


Figura 3.5 - Captură ecran în timpul jocului Agar

3.4.2. Slither

Un alt joc multiplayer masiv de acțiune, similar, este Slither [8]. În acest joc, jucătorul controlează un șarpe. Scopul jocului este de a colecta hrană și de a crește șarpele, omorând alți șerpi, iar mai apoi devorându-i și pe aceștia.

Jocul are o interfață grafică foarte dezvoltată și atrăgătoare, ceea ce din păcate însă duce la întârzieri la randare, care la rândul lor provacă întârzieri la comunicarea cu serverul, iar experiența utilizatorilor are foarte mult de suferit din aceste motive.

Flib dorește câștigarea atenției utilizatorilor prin posibilitățile de a interacționa jucătorii într ei și prin dorința de competiție. Interfața grafică se preferă a fi simplă, asemănătoare cu cea a jocului Agar pentru ca oricine să poată juca, fără a fi nevoie de un calculator foarte performant.

Figura 3.6 - Captură ecran în timpul jocului Slither

3.4.3. Diep

Diep [9] este un joc în care scopul este ca fiecare jucător să își construiască propriul tanc și să elimine oponenții. Pentru a-și dezvolta tancul, utilizatorul trebuie să distrugă anumite obiecte plasate în mod aleator pe harta de joc. La un anumit număr de obiecte distruse, utilizatorul are posibilitatea de a face upgrade la o anumită caracteristică.

Interfața grafică a jocului este simplă, dar din păcate jocul consumă foarte multă memorie, browserul ajungând la limita maximă se va bloca.

Spre deosebire de Diep, în Flib s-a dorit ca resursele colectate să aibă efect imediat asupra caracteristicilor jucătorului.

Figura 3.7 - Captură ecran în timpul jocului Diep

3.4.4. Analiză comparativă

În continuare se va preznta o comparație între sistemul Flib și celelalte trei jocuri similare: Agar, Slither, Diep.

Criteriu de evaluare	Flib	Agar	Slither	Diep
Permite mod multiplayer	Da	Da	Da	Da
Aplicație WEB	Da	Da	Da	Da
Interfață intuitivă	Da	Da	Da	Da
Oferă top al jucătorilor	Da	Da	Da	Da
Componente de inteligență artificială	Da	Nu	Nu	Parțial
Permite modificarea caracteristicilor jucătorului	Da	Nu	Nu	Da
Minimap	Nu	Nu	Da	Da
Coordonatele poziției curente	Da	Nu	Nu	Nu
Oferă tutoriale	Da	Da	Nu	Nu

Tabel 3.1 - Analiză comparativă a jocurilor

În tabelul 3.1 pot observa o serie de similaritati între aceste sisteme, dar și diferențe apărute datorită cerințelor specifice fiecărui joc și modului de a fi jucate. Se poate observa că jocul Flib nu are implementat un minimap care să ofere jucătorului informații despre pozițiile celorlalți jucători, dar ca și element de localizare se afișează coordonatele carteziene care sunt folositoare atunci când un grup de prieteni vor să se întâlnească să lupte împreună.

Capitolul 4. Analiză şi Fundamentare Teoretică

În acest capitol va fi prezentata atât o analiză și diferite aspecte generale fundamentate teoretic cât și cazurile de utilizare ce se propun a fi implementate în acest sistem. Vom începe prin a prezenta cerințele atât funcționale cât și cele nonfuncționale, urmând ca mai departe să fie descrise tehnologiile care au fost folosite pentru implementarea lor.

4.1. Cerințe funcționale

Cerințele funcționale reprezintă descrierea comportamentului sistemului în anumite circumstanțe și posibilitățile utilizatorului de a interacționa cu acesta.

În tabelul de mai jos vor fi descrise cerințele funcționale ale acestui sistem. Aceste cerințe nu reprezintă o versiune finală a aplicației, ele putând fi completate, modificate sau îmbunătățite în viitor. Unele dintre ele au fost implementate, altele au rămas neimplementate nefiind considerate suficient de importante, dar putând fi implementate ulterior. Nivelul de importanță al fiecărei cerințe reprezintă, pe o scară de la 1-5, cât de necesară este implementarea cerinței respective. Nota 5 reprezintă faptul că fără acea componentă jocul nu este posibil, fiind strict necesară implementarea ei, iar la polul opus, nota 1 reprezintă nivel foarte scăzut de importanță care nu afectează jocul.

Criteriul cerinței	Identificator	Descrierea cerinței funcționale	Nivel de importanță	Implementat
Elemente importante ale jocului	CF 1.1	Joc multiplayer real-time online	5	Da
	CF 1.2	Generare aleatoare de resurselor pe harta de joc	5	Da
	CF 1.3	Colectarea resurselor pentru creșterea în masă	5	Da
	CF 1.4	Atacul altor jucători pe baza resurselor colectate	4	Da
	CF 1.5	Jucătorul atacat pierde resurse dacă este lovit	4	Da
	CF 1.6	Distrugerea inamicului dacă masa sa este sub o anumită limită	4	Da
	CF 1.7	Mișcarea jucătorului pe harta de joc	5	Da
Inteligență artificială	CF 2.1	Crearea unui bot folosind inteligență artificială	3	Da
	CF 2.2	Bot-ul atacă automat jucătorii de pe o anumită rază	3	Da
	CF 2.3	Bot-ul colectează resurse singur	3	Da
	CF 2.4	Bot-ul poate fi distrus	3	Da
Autentificare	CF 3.1	Posibilitate de autentificare	2	Da
	CF 3.2	Posibilitate de înregistrare cu facebook	2	Nu

Criteriul cerinței	Identificator	Descrierea cerinței funcționale	Nivel de importanță	Implementat
	CF 3.3	Posibilitatea de a selecta un nume care va fi afișat ulterior în joc	4	Da
	CF 3.4	Posibilitatea utilizatorilor înregistrați de a-și alege culoare favorită	1	Nu
	CF 3.5	Posibilitatea utilizatorilor înregistrați de a alege o cameră la care să se conecteze	2	Nu
Extra	CF 4.0	Afișarea instrucțiunilor de joc	3	Da

Tabel 4.1 - Descrierea cerințelor funcționale

Prima grupă de cerințe funcționale se referă la funcționalitățile jocului. Pentru a oferi utilizatorilor posibilitatea de a juca multiplayer este nevoie de o conexiune, prin WebSockets, stabilă între server și client. Subpunctele primei grupe se referă la funcționalități secundare din această grupă. Pentru generarea și afișarea resurselor se va folosi un generator de numere random și un QuadTree în care le vom stoca. Alegând structura de date QuadTree timpul de căutare în spațiul 2D este O(log n). Pentru colectarea de resurse, se va verifica poziția jucătorului pe hartă, iar dacă circumferința sa atinge circumfernița unei resurse, masa acesteia va fi adăugată la masa jucătorului. Următoarele subpuncte se referă la partea de atac al oponenților întâlniți pe parcursul jocului.

A doua grupă se referă la crearea unui jucător virtual (bot) care să simuleze jucători reali în cazul în care numărul de jucători reali este mic într-o anumită cameră. Pentru implementarea acestor boti se va crea o mini aplicație web care să genereze un anumit număr de boți. Bot-ul va trebui să fie în stare să atace singur alți playeri, să se ferească și să colecteze resurse.

A treia grupă se referă la funcționalitatea de înregistrare și autentificare a unui utilizator nou. Cu ajutorul API-ului de la Facebook utilizatorul va putea să își creeze cont instant, stocând în baza de date a sistemului datele jucătorului și un token pentru recunoașterea sa în viitor. Celelalte subpuncte reprezintă avantaje care le-ar putea avea un jucător autentificat.

A patra grupă o reprezintă elementele care nu sunt legate de joc în mod direct, ci sunt funcționalități adiționale care oferă jucătorului informații despre cum se desfășoară un joc.

4.2. Cerințe nonfuncționale

Cerințele funcționale trebuie luate în considerare cât mai timpuriu în procesul de proiectare, deoarece spre deosebire de cerințele funcționale, cele nonfuncționale sunt impuse sistemului, sunt constrangeri care trebuiesc satisfăcute. Cu cât sunt detectate și luate în considerare mai târziu aceste cerințe, cu atât există riscul mai mare de fi necesare modificări ulterioare în diferite module ale aplicației, sau chiar de refacerea totală a anumitor funcționalități.

Cerințe funcționale	Descriere
Scalabilitate	Scalarea se poate face pe o singura axa, prin copierea mai multor instanțe pe mai multe noduri. Împărțirea pe mai multe noduri se va face atunci cand procesorul nu mai poate face fata, sau numarul de utilizatori conectati este mai mare de 30.
Securitate	Securitatea este un factor critic în orice sistem, utilizatorii având încredere că datele lor rămân confidențiale. Jucatorii inregistrati vor avea parolele criptate in baza de date.
Performanță	Performanța se referă la timpul de răspuns necesar sistemului pentru a procesa o cerere din partea unui utilizator. S-a pus foarte mare accent pe implementarea unui sistem care comunică în timp real cu clientul. Pentru a reduce viteza de răspuns a sistemului și pentru a da senzația că jocul este realtime s-a ales structura de date QuadTree și s-au implementat algoritmi în așa fel încât utilizatorul să primească detalii despre oponenții și resursele care sunt în jurul său la o distanță anume.
Portabilitate	Aplicația rulează în browser, iar datorită acestui fapt se va face abstracție de sistemul de operare. Cerința minimă este ca utilizatorul să folosească un browser modern cu suport HTML5. La partea de backend, fiind un server NodeJS, la fel cerințele sunt minime, sistemul de operare nu contează.

Tabel 4.2 - Descrierea cerințelor nonfuncționale

4.3. Tehnologii folosite

4.3.1. Node.js

Node.js [10] este un famework open-source compatibil cu orice sistem de operare fiind folosit pentru dezvoltarea părții de server a aplicațiilor Web. Chiar dacă Node.js nu este un framework Javascript, majoritatea modulelor sunt scrise in Javascript, iar programatorii pot scrie module noi tot în acest limbaj. Runtime-ul intepretează codul scris în javascript folosind engine-ul Javascript V8 de la Google.

Acest framework conține intern o bibliotecă pentru server HTTP ceea ce face posibilă rularea unui server de web, fără utilizarea unui software extern, permițând astfel un control mai mare asupra felului în care serverul intern funcționează. Pe scurt, folosindu-ne de această librărie, putem crea aplicații interactive, cu comunicare în timp real, fără a suprasolicita serverul sau clientul cu diferite request-uri inutile și fără a face trafic în excess, așa cum s-ar întâmpla dacă am opta pentru alternativa AJAX. Un avantaj foarte mare al Node.js este că poate rula în paralel mai multe cereri chiar dacă cea inițială nu a fost finalizată încă.

Node.js conține o arhitectură event-driven capabilă de I/O asincron [11]. Acest design intenționeaza optimizarea scalabilității în aplicațiile web care efectuează un numar

mare de operații de input/output, si mai ales aplicațiilor care necesită comunicare real-time (ex. Jocurile in browser).

Avantajul principal al alegerii acestei tehnologii este limbajul Javascript, care se dorește a fi principalul și singurul limbaj de programare folosit atât pentru partea de server side cât și pentru partea de client, astfel orice dezvoltare ulterioară necesită cunoștințe doar ale acestui limbaj. De asemenea un alt avantaj este comunicarea prin socket a mesajelor de tip JSON care sunt native Javascript, putând fi capturate și folosite ca obiecte fără a fi necesară o mapare specială. Tot în același sens, implementarea cu baza de date MongoDB este foarte ușoară deoarece obiectele stocate în ea sunt tot de același tip.

4.3.1.1. Npm

Npm [12] este un manager de pachete scris în limbajul Javascript care vine împreună la instalare cu mediul de rulare Node.js. Este format dintr-un client cu linie de comandă care interacționează cu un registru remote. Npm permite utilizatorilor să consume sau să distribuie module Javascript care sunt disponibile în registru. Toate modulele din registru conțin un fișier JSON care include date despre modulul respectiv. Pachetele stocate în npm nu țin cont de cine a le-a stocat, se bazează pe principiul primul venit - primul servit, ceea ce înseamnă că dacă cineva își va șterge pachetul înregistrat, oricine altcineva va putea să înregistreze un pachet cu același nume și să cauzeze daune tuturor celor care au folosit pachetul respectiv. De asemenea pentru publicarea unui pachet nu există un process de verificare a conținutului, ceea ce înseamnă ca pachetele pot să fie nesigure sau malițioase.

Chiar dacă nu se poate verifica dinainte cât de sigur și de calitate este un anumit pachet, npm oferă statistici despre cât de folosit și câte dependințe are acesta, făcând-ul un tool foarte puternic și foarte folositor pentru managementul dependințelor proiectelor.

O alternativă a folosirii NPM ar fi fost package manager-ul ”Yarn” [13], care oferă o gamă largă de librării, cu o performanță și securitate ridicată. Chiar dacă Yarn are avantajele sale, numărul de librării este într-un număr mult mai redus față de NPM. Astfel am ales NPM pentru două avantaje: numărul mare de dependințe din care se poate alege și integrarea nativă cu Node.JS.

4.3.2. Express.js

Express.js [14] este un framework care rulează pe platforma Node.js pentru o variate larga de aplicații web. Acest framework are ca scop de a răspunde diferitelor cereri http ce vin de la browser-ul clientului.

Unul din beneficiile pe care le aduce Express.js este că poate să intercepteze diferite apeluri și să facă routing cu scopul de a oferi clientului datele necesare în funcție de pagina pe care o accesează. De asemenea există posibilitatea de a servi un folder static care să conțină toate informațiile necesare utilizatorului.

Express.js poate interacționa cu alte module externe Node.js [15] care pot fi utile în momentul în care dorim să scriem un document HTML într-un limbaj care să poată fi

compilat pe server înainte de a fi servit clientului. Jade [16] este un astfel de limbaj care poate fi compilat și trimis la client, dar el va fi discutat într-un alt subcapitol.

Un alt aspect important prezent în Express.js este dat de funcția middleware care este o funcție anonimă care are încorporată ca parametrii obiectele req, res dar și next, next fiind folosită pentru a începe rularea următoarei funții anonime.

Avantajul principal al alegerii Express.js este comunitatea extrem de mare care o are, fiind cel mai mare și mai folosit framework pentru Node.js. Oferă căi foarte simple de a porni un server și promovează refolosirea codului cu ajutorul componentei de routing încorporată. Câteva alternative ar fi fost ”Koa” și ”Hapi” dar fiecare are părțile ei negative care nu s-ar fi potrivit în acest proiect. Koa este instabilă, iar cea de-a doua, Hapi, are lipsuri care ar fi îngreunat developmentul, fiind gândită în special pentru aplicațiile mai mari.

4.3.3. Simple-Quadtree

Simple-Quadtree [17] este o implementare minimală a unei structuri de date de tip arbore cu patru ramuri care suportă operațiile put, get, remove, clear pe obiecte 2d care au coordonate x, y și dimensiuni w, h.

Această librărie este compatibilă cu toate browserele, dar este des folosită pe partea de server fiind disponibilă ca pachet în cadrul managerului de pachete al Node.js.

Pentru a putea fi folosit se va instanția un obiect de tip QuadTree oferindu-se ca parametrii limite ale acestuia. După ce QuadTree-ul a fost inițializat, se vor putea adăuga orice tip de elemente (obiecte) cu restricția ca ele să conțină coordonatele carteziene x, y și dimensiunile w și h ca și numere pentru a indica suprafața obiectului. După ce s-a populat acest arbore se vor putea extrage și modifica elemente cu ajutorul funției get. Dacă se dorește ștergerea unui element din arbore, se va folosi funcția delete. În momentul în care developer-ul consideră că nu mai are nevoie de elementele din quadtree, el va putea folosi funcția clear care va curăța arborele de toate elementele care au fost introduse.

În următorul subcapitol se va detalia structura de date quadtree și algoritmii folosiți de acesta.

4.3.3.1. Quadtree

Quadtree [18] este o structură de date în care fiecare nod intern are exact patru ramuri considerate copii. Quadtree este des întâlnit în partiționarea unui spațiu cu două dimensiuni prin subdivizarea recursivă în patru cadrane sau regiuni. Aceste regiuni pot avea diferite forme arbitrare, dar de obicei sunt pătrate sau dreptunghiuri. Există structuri de date similare, dar toate au câteva caracteristici comune:

- Descompun spațiul în celule adaptabile
- Fiecare celulă are o capacitate maximă. Când capacitatea maximă a fost atinsă, celula se împarte în alte patru celule

Quadtree-urile se pot clasifica în funcție de tipul de date pe care îl reprezintă, incluzând arii, puncte, linii sau curbe. Câteva tipuri mai comune sunt: quadtree de

regiune, quadtree pentru puncte, quadtree pentru poligoane, etc.

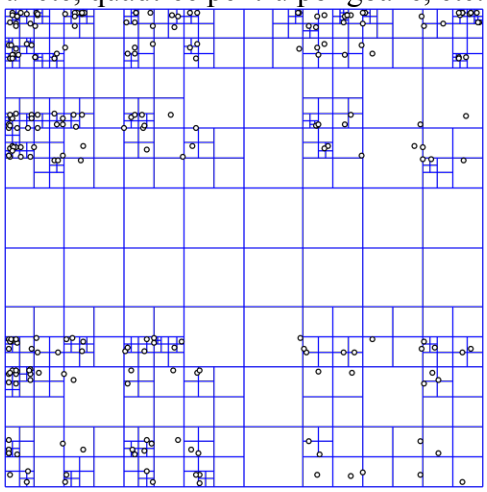


Figura 4.1 - Reprezentarea grafică a unui quadtree pentru puncte[1]

Avantajul acestei structuri de date este că oferă rezultate foarte bune, fiind foarte optim pentru căutarea obiectelor pe suprafețe 2D. Alternativele ar fi fost căutarea obiectelor stocate într-o structură de date de tip array sau tree, care ar fi fost foarte lente în comparație cu Quadtree, fiind necesară o parcurgere mult mai adâncă.

4.3.4. CreateJS

CreateJS [19] este o suită de librării modulare și instrumente care funcționează împreună sau independent pentru a crea aplicații bogate în conținut interactiv folosind tehnologiile web cu suportul instrumentelor din HTML5. Librăriile CreateJS sunt interoperabile cu toate aplicațiile de browsing atât desktop cât și mobile moderne, și au fost testate astfel încât să obțină performanțe extraordinare în toate acestea.

CreateJS este format din patru librării principale care vor fi prezentate în continuare:

4.3.4.1. EaselJS

EaselJS [20] oferă soluții pentru dezvoltarea aplicațiilor care folosesc interacțiune și elemente grafice cu ajutorul elementului Canvas din HTML5. Oferă un API care este foarte asemănător cu cel din Adobe Flash, incluzând liste ierarhice a elementelor de afișat.

4.3.4.2. TweenJS

TweenJS [21] este o librărie care generează cadre intermediare pentru a fi folosite în JavaScript. A fost dezvoltat pentru a se integra cu EaselJS, dar nu este dependentă de acesta. Oferă suport pentru generarea cadrelor intermediare atât pentru obiecte numerice cât și pentru proprietăți CSS.

[1] https://en.wikipedia.org/wiki/Quadtree

4.3.4.3. SoundJS

SoundJS [22] este o librărie care oferă posibilitatea introducerii sunetelor cu ajutorul componente WebAudio din HTML5 sau Flash folosind un plugin. Deoarece nu toate browserele oferă suport pentru WebAudio sau Flash, SoundJS selectează automat componenta care va funcționa pentru browserul utilizatorului.

4.3.4.4. PreloadJS

PreloadJS [23] oferă posibilitatea de a încărca diferite asset-uri, incluzând imagini, sunete, cod Javascript, date în format text, etc. În mod implicit, PreloadJS se foloseste de funcția XHR2 unde browserul suportă, iar în caz contrar se vor folosi alte metode. Un punct forte este că acceptă mai multe cozi de asset-uri și conexiuni în paralel.

4.3.5. Jade

Jade [24] este un template engine de mare performanță influențat de Haml și implementat cu Javascript pentru Node.js și browsere. Este un sistem care combină diferite template-uri pentru a genera pagini web cu posibilitatea de a injecta date prezente în mediul de rulare Nodejs. Template-urile pot fi folosite pentru a genera componente mici, pagini statice și chiar pagini dinamice care se modifică în funcție de cerere.

Un template Jade este format din:

- Un template engine - principalul element de procesare al sistemului
- Resurse pentru conținut: orice tip de resurse (de exemplu: date dintr-o bază de date), fișiere xml, stream-uri de date sau alte tipuri de date din rețea
- Resurse șablon: șabloane web specifice limbajului Jade

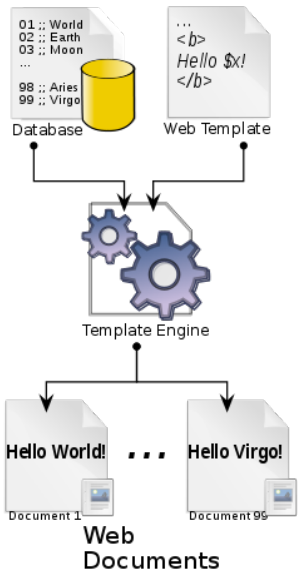


Figura 4.2 - Exemplu de procesare pe partea de server a unui template[2]

[2] https://en.wikipedia.org/wiki/Web_template_system

Unul dintre scopurile folosiri unui astfel de motor este de a crea aplicații cât mai flexibile și ușor de întreținut. Pentru a atinge acest scop trebuie luată în considerare separarea conceptelor de business logic față de conceptele de presentation logic. Programatorii folosesc sisteme de șablonare pentru a menține această separare.

4.3.6. HTML

HTML [25] (Hyper Text Markup Language) este un limbaj de markup folosit în descrierea structurii paginilor web. Documentele HTML sunt descrise cu ajutorul tagurilor specifice limbajului. Fiecare tag specifică conținut diferit, astfel putând fi posibilă introducerea imaginilor, a scrisului ingrosat, a marca sfarsitul liniei, etc.

Structura tag-urilor specifice limbajului este inserarea unui cuvânt, descriptiv și predefinit, între paranteze unghiulare. În general tag-urile vin în perechi pentru a marca începutul și sfârșitul unui bloc. Primul tag este de regulă tag-ul de start, iar cel de-al doilea tag este cel de final,care marchează terminarea unei particularități asupra unei componente. Exceptie de la regulă o fac tag-urile al căror conținut nu ar fi relevant în rezolvarea unei particularități.

Datorită acestor taguri, browserul le interpretează ca și diferite componente și le afișează în ordinea scrierii lor. Browserul nu va afișa tag-urile, dar le va folosi continutul pentru a afisa pagina. Compoentele sunt afișate diferit în funcție de tag, fiecare având propria implementare în browser. Pentru a modifica modul de afișare al componentelor se pot încărca în pagină diferite stiluri custom numite și Cascading Style Sheets. Acestea pot modifica dimensiunile fontului, culoarea textului, distanțe între elemente, etc. De asemenea pentru a modifica comportamentul componentelor web se pot adăuga și scripturi scrise în limbajul Javascript care să facă pagina mai interactivă, ca și exemplu schimbarea unei imagini la apăsarea unui buton.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

Figura 4.3 Exemplu de cod HTML

4.3.6.1. HTML5

Conform W3Schools HTML5 [26], a fost publicat în octombrie 2014 pentru a îmbunătăți limbajul oferind suport pentru multimedia. Așadar au fost introduse nativ diferite caracteristici sintactice oferind posibilitatea introducerii continutului grafic si

multimedia in pagina, dar de asemenea si elemente folosite pentru îmbogățirea conținutului semantic. Pentru partea multimedia s-au introdus tag-urile **video**, **audio** si **canvas.** De asemenea s-a introdus suport pentru grafică vectorială scalabilă (SVG) și MathML pentru formule matematice. Pentru îmbunătățirea conținutului semantic al documentelor, s-au introdus structurile **main**, **section**, **article**, **header**, **footer**, **aside**, **nav** si **figure**. Anumite atribute prezente in HTML versiunea a patra au fost sterse, modificate, redefinite sau standardizate.

Cu toate aceste modificări HTML5 s-a reușit includerea unui model detaliat de procesare pentru a încuraja dezvoltarea aplicațiilor pe mai multe platforme.

Figura 4.4 Exemplu player video HTML5

4.3.6.2. Canvas

Canvas [27] este elementul HTML folosit pentru desenarea de elemente grafice cu ajutorul unui script (de obicei JavaScript). Elementul în sine este doar un container gol peste care se pot desena text, linii, cercuri, poligoane, imaginii etc.

```
<canvas id="game" width="500" height="500" >
        Browser-ul tău nu suportă tag-ul canvas.
</canvas>
```

Figura 4.5 Exemplu de descriere a unui canvas simplu

```
var canvas = document.getElementById("game");
var ctx = canvas.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(100,50);
ctx.stroke();
```

Figura 4.6 Exemplu de desenare a unei linii canvas

4.3.7. CSS

Cascading Style Sheets (CSS) [28] este un limbaj de stilizare a elementelor web scrise într-un limbaj de markup. În ziua de azi în stilizarea paginilor web, CSS-ul este sigurul limbaj de stilizare folosit. Cu ajutorul acestui limbaj se pot seta atât culori, fonturi, etc. ale textului, cât și poziționa elementele pe pagina web. Există două metode de a incorpora CSS-ul în pagină și acestea sunt fie încorporate direct ca și conținut al tag-ului style, fie ca și fișier extern css.

Ca și alternativă scrierii CSS-ului, se mai pot folosi și limbaje mai moderne LESS sau SASS care vor folosi ulterior compilatoare care vor transforma codul respectiv în CSS pentru a putea fi interpretat de browser. Acestea, chiar dacă sunt foarte asemănătoare, aduc multe îmbunătățiri. Exemple de beneficii ar fi stabilirea anumitor constante, variabile și funcții. După compilarea codului, acestea sunt transformate în CSS pur și nu vor mai exista.

Sintaxa CSS (exemplul 4.7) este formată întotdeauna dintr-un cap care conține descrierea elementului și un corp care descrie cum trebuie să arate toate elementele din HTML descrise de capul acesteia.

```
span {
        font-family:Arial;
        font-size:10px;
        color: #FFF;
}
```

Figura 4.7 Exemplu de cod CSS

În momentul aplicării schimbăriilor asupra HTML-ului, CSS-ul va crea o schemă de priorități pentru a determina care stil are cea mai mare importanță și care trebuie aplicat. În acest caz cele mai prioritare stiluri sunt cele care sunt descrise cel mai mult în cap.

4.3.8. Javascript

Javascript [29], dezvoltat inițial de NetScape, în curent împreună cu fundația Mozilla, este un limbaj de programare dinamic și interpretat, care ajută la introducerea unor funcționalități în plus paginilor Web. Împreună cu CSS și HTML, cele trei tehnologii sunt considerate baza internetului, majoritatea website-urilor având partea de frontend scrisă în aceste limbaje.

Chiar dacă există multe similarități între Java și JavaScript, acestea nu trebuie confundate. Ele au fost concepute total diferit și în alte scopuri. Javascript a fost standardizat in specificatiile limbajului ECMAScript.

În prezent Javascript nu este folosit doar pe partea de Web, ci și în diferite documente PDF, widget-uri desktop și chiar și in dezvoltarea sistemelor de backend cu ajutorul Node.js. Este foarte des folosit de asemenea în dezvoltarea jocurilor fiind compatibil pe toate platformele inclusiv pe telefoanele mobile, aceasta fiind una dintre

cele mai mari avantaje care nici un alt limbaj nu le mai are.

Javascript conține 6 tipuri de date: boolean, null, undefined, number, string și object. Array-ul este tot un obiect în care cheile sunt obținute din lungimea șirului.

Codul javascript este executat local în browser-ul utilizatorului astfel poate răspunde cu ușurință la acțiuni și evenimente. Javascript-ul este într-o strânsă legătură cu DOM-ul [30] astfel că are access la toate elementele din pagină și evenimente lansate de browser, astfel putând intercepta diferite taste apăsate, click-uri ș.a.m.d, oferindu-i puterea de a ține o parte de logică în browserul utilizatorului. De asemenea este posibilă și interacțiunea cu server-ul prin AJAX [31], permițându-i utilizatorului să facă requesturi HTTP în background.

4.3.8.1. EcmaScript 2015

EcmaScript 2015 (ES6) [32] este un standard care definește limbajul de programare JavaScript. Versiunea anterioară EcmaScript 5, lansată oficial în 2009, a adus foarte multe îmbunătățiri limbajului, dar a lăsat și goluri care făceau ca limbajul să pară puțin inferior altor limbaje. ES6 a introdus caracteristici pe care dezvoltatorii le așteptau de foarte mulți ani, transformând limbajul într-o unealtă foarte puternică. Printre îmbunătățirile majore se află și module, clase, funcții arrow, promises, și multe altele.

În momentul de față, browserele au suport nativ pentru ES6, dar din păcate, nu toată lumea are access la un browser compatibil și din această cauză nu vor putea beneficia de îmbunătățirile aduse în această versiune. Totuși o soluție pentru programatori sunt așa numitele transpilers care transformă codul ES6 în cod ES5. Cel mai cunoscut asemenea transpiler este Babel.

ES6 a adus multe îmbunătățiri, majoritatea vor face programatorilor viața mai ușoară și îi va ajuta să scrie cod mai curat și mai lizibil, dar îi va și obliga să gândească puțin diferit față de cum a u fost obișnuiți.

4.3.9. JSON

JSON (Javascript Object Notation) [33] este un standard conceput pentru schimbul de date dintre diferite servicii online. A fost conceput ca și o alternativă pentru XML fiind mult mai ușor de generat și de înțeles de către oameni (figura 4.8). Este complet independent de limbajul de programare Javascript chiar dacă derivă din el. În general este folosită pentru serializarea și trimiterea de date structurate între diferite sisteme deoarece majoritatea limbajelor de programare au implementat nativ parsarea și generarea obiectelor de acest tip.

Formatul JSON suporta tipuri de date astfel fiind ușor de folosit în diferite cazuri:

Number - echivalentul tipului de date double, pot fi folosite numere cu virgulă

String - un șir de caractere scris între ghilimele duble

Boolean - true sau fals

Array - O secvență ordonată de valori

Value - un string, un număr, true sau false, null, etc

Object - o colecție neordonată de perechi cheie : valoare

Whitespace - poate fi folosit între orice pereche de tokeni
null - reprezinta lipsa datelor

```
{
        "id": 1,
        "prenume" : "Sergiu",
        "varsta": 23,
        "casatorit": false
}
```

Figura 4.8 Exemplul unui obiect JSON care descrie o persoana

4.3.10. MongoDB

MongoDB [34] este o bază de date, concepută de compania 10gen, care face parte din familia sistemelor de NoSQL. Diferența fața de bazele de date relaționale este că stocarea datelor nu se face în tabele, ci le stochează sub formă de documente JSON cu scheme dinamice. Fiecare obiect JSON poate fi stocat în baza de date fără să intereseze forma acestuia.

MongoDB a fost concepută ca și o bază de date open-source scrisă în C++. Sistemul poate conține mai multe baze de date, colecții sau indecși. Faptul că nu există o schemă pentru câmpurile dintr-un document poate duce de multe ori la anumite inconsistențe de aceea se obișnuiește ca o coleție să aibă o structură omogenă. Există câteva caracteristici ale MongoDB care trebuie luate în considerare:

- Stocarea datelor sub formă de documente - reduce nevoia de join, obiectele sunt stocate în așa fel încât să conțină toate informațiile necesare despre obiectul respectiv.
- Prezintă suport pentru indexare
- Auto-Sharding - partitionarea datelor pe orizontală se face automat pe orizontală, citirile și scrierile fiind distribuite pe partiții, iar lipsa joinurilor face ca interogările să fie mai rapide
- Limbajul de interogare - păstrează principii SQL și C++ fiind ușor îmbunătățit
- Map Reduce - folosește map/reduce pentru agregare

In figura 4.8 este prezentată o paralelă între MySQL și MongoDB, aceasta evidențiind asemănările dintre cele două baze de date.

Termeni MySQL	Concepte MongoDB
Bază de date	Bază de date
Tabelă	Colecție
Index	Index
Rând	Document BSON
Coloană	Câmp BSON
Join	Încapsulare și legatură
Cheie primară	Câmpul cheie unică _id
Group by	Agregare

Figura 4.8 - Paralelă MySQL vs MongoDB

Avantajele principale folosirii acestei tehnologii sunt: salvarea obiectelor sub formă de JSON și faptul că nu sunt necesare relații între tabele. MongoDB este de altfel mult mai rapdiă decât baza de date MySQL la citire, ceea ce oferă un alt avantaj luat în considerare.

4.3.11. GIT

GIT [35] este un sistem de versionare al codului. Acest software oferă posibilitatea de a face management al modificărilor din documente, cod sursă al diferitelor aplicații în orice limbaj de programare și orice alte colecții de informații. Fiind un software care gestionează versiunile, fiecare versiune a fișierelor este identificată cu hash specific versiunii. Fiecare revizuire este asociată cu persoana care a publicat modificările și timpul la care au fost publicate schimbările. GIT oferă posibilitatea de organizare într-un mod logic pe mai multe ramuri a schimbărilor.

Este un tool foarte puternic datorită faptului că în momentul în care schimbările au fost comise, ele nu vor mai putea fi șterse, astfel programatorul nu își va putea pierde munca cu ușurință. De asemenea oferă posibilitatea de a crea ramuri noi, de a aduce schimbările de pe alte ramuri, de a reveni la versiuni anterioare, ș.a.m.d.

Git de asemenea oferă posibilitatea de a lucra în echipă, fiind posibilă conectarea cu un server pe care se va seta un repository la care să se poată conecta mai mulți developeri. Fiecare utilizator își va face setup local care se va conecta la repository-ul respectiv și va începe prin a face schimbări local, a commite schimbările, iar în momentul în care va dori publicarea versiunii noi, el va face push. Metoda de push va plasa schimbările pe server, astfel toti utilizatorii vor avea acces la aceste schimbări.

În continuare se vor prezenta câteva comenzi importante și des folosite:

git init - creaza un repository nou

git clone - copiaza un repository existent

git fetch - ia schimbările din repository-ul remote dar nu le aplică local

git pull - ia schimbările din repository-ul remote și le aplică local

git branch - crează un branch local cu baza în punctul din care s-a executat

comanda

git add - pregătește schimbările pentru a fi comise
git commit - comite schimbările efectuate în cod
git push - împinge un anumit branch cu commit-urile făcute către server

Unul dintre dezavantajele git-ului este că nu oferă o interfață grafică, iar utilizatorilor începători le este greu să se acomodeze. În următorul subcapitol va fi prezentată o alternativă care oferă accesul la o interfață grafică, ușor de folosit atât de cei începători cât și de cei experimentați.

4.3.11.1. SourceTree

SourceTree [36] este un software, dezvoltat de compania BitBucket, care oferă o interfață grafică pentru GIT. SourceTree implementează toate funcțiile disponibile în git. SourceTree ajută la managementul repository-urilor atât locale cât și al celor remote printr-o interfață ușoară de înțeles și celor începători. Este disponibil însă doar pe sistemele Windows și Mac. Figura 4.9 reprezintă o ilustrare a interfeței grafice oferite de acest software.

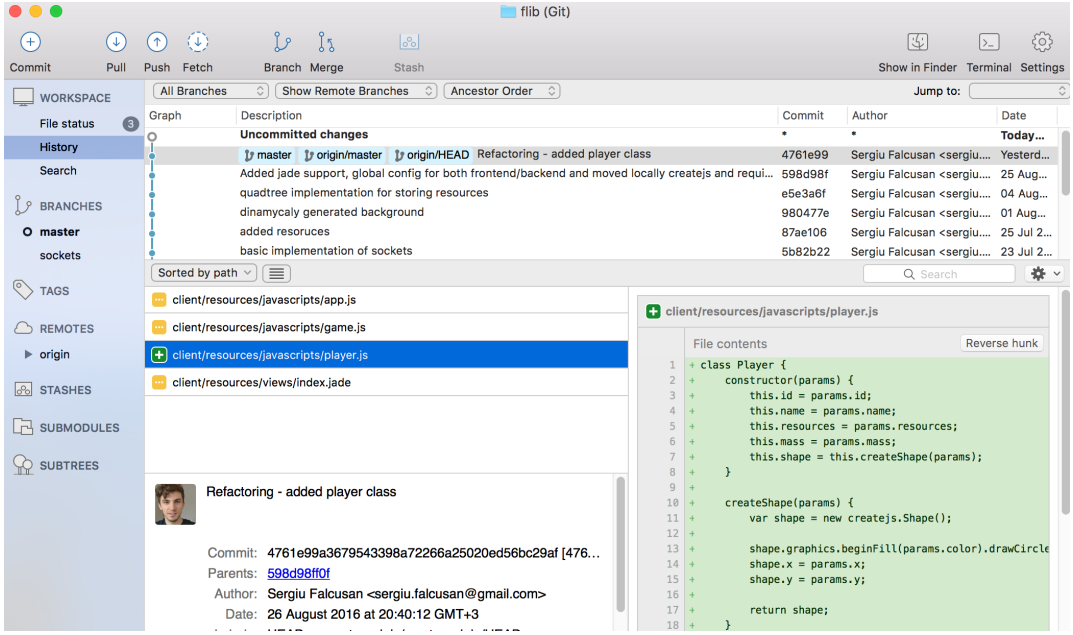


Figura 4.9 Interfața grafică a programului SourceTree

4.4. Principii de dezvoltare

O strategie bună de a începe orice proiect este de a se impune anumite principii de la bun început care să fie urmate de toți contribuitorii. Pentru a crea un software bun este nevoie de disciplină care implică: concentrare, prezență de spirit și gândire. Aplicarea principiilor poate părea de multe ori o pierdere de timp, dar pe lungă durată se va observa o mai mare flexibilitate și o ușurință în dezvoltarea softului.

4.4.1. Design patterns

La dezvoltarea proiectelor software, de-a lungul timpului, s-au observat anumite probleme comune în design. Pentru majoritatea problemelor de acest gen au apărut și soluții care prezintă un template posibil de integrat în anumite aplicații. Aceste template-uri nu sunt bazate pe un limbaj de programare și nu este un cod de bază de la care se pornește o aplicație, ci este un concept care face parte din domeniul modulelor și interconexiunilor dintre anumite componente. În următoarele rânduri se vor prezenta câteva design pattern-uri folosite în scopul implementării acestui proiect.

4.4.1.1. Singleton

Conceptul de singleton în proiectare se referă la instanțierea unei clase o singură dată, mai apoi fiind refolosită de fiecare dată oriunde este necesară în proiect. Este foarte importantă în cazul serviciilor care trebuie să proceseze anumite informații.

Implementarea acestui pattern se face prin crearea unei noi instanțe a clasei dacă aceasta nu există, iar dacă există atunci se întoarce referința către obiectul existent.

Acest pattern este cel mai întâlnit în cadrul proiectelor Node.js datorită faptului că metoda ”require” care importă anumite clase instanțiază o singură dată un obiect, iar apoi folosește referința sa de fiecare dată când se reimportă acea clasă.

4.4.1.2. Observer

Observer pattern are scopul de a crea relații de la un obiect la mai multe în scopul informării obiectelor conectate despre modificări efectuate în interiorul său. În acest proces este implicat un obiect denumit ”subiect” și o listă asociată de obiecte dependente, denumite ”observatori”.

4.4.2. Clean code

Mai multe lucrări și articole științifice relatează despre diferite moduri de a scrie codul corect și cum se poate structura astfel încât pe viitor să se evite problemele care să necesite refactoring. Un cod scris corect de la bun început, este un cod mai ușor de citit, mai ușor de înțeles și mai ușor de dezvoltat ulterior. Chiar dacă în teorie sunt multe lucrări scrise pe această temă, o definiție a codului curat nu se poate da, dar în următoarele paragrafe se vor evidenția câteva principii care sunt aplicate în mod general.

Conform [37] codul curat este elegant, simplu, eficient, direct, clar, inteligent, citibil de alte persoane, fără surprize, dependințe puține și explicite, conține teste automate, minimizează numărul de clase și metode, tratează erori, nu conține nimic evident care ar putea fi modificat ușor de alte persoane și ilustrează faptul că autorului i-a păsat când l-a scris. Codul poate fi recitit de multe ori, așa că este important ca atunci când altcineva vrea să mai adauge cod să-l poată înțelege ușor.

O principală cauză care provoacă greșeli sunt denumirile claselor sau al metodelor acordate într-un mod incorect. Numele trebuie să fie semnificative, să dezvăluie intenția și să fie ușor de pronunțat. Numele claselor trebuie să fie substantive, iar numele metodelor să fie expresii care să conțină un verb. Se recomandă folosirea unui cuvânt

care să descrie un concept în mod consinsent.

Metodele pentru a fi cât mai curate trebuie sa fie cât mai scurte și să facă doar un singur lucru. Pentru aceasta trebuie evitată scrierea blocurilor de cod care nu fac ceea ce numele funției descrie. Fiecare bloc de cod trebuie introdus într-o metodă care face exact ceea ce îi descrie numele cât mai abstract și general posibil. Ordinea funcțiilor trebuie să fie făcută în ordinea în care ele sunt apelate, astfel citirea codului să poată fi de sus în jos.

Comentariile considerate bune sunt cele informative care explică intenții, avertizări ale consecințelor, todo-uri, marcare a codului imporant și documentații ale API-ului public. De evitat sunt comentariile redundante, comentariile derutante, comentariile folosite în schimbul scrierii unei metode separate, codul comentat, informații irelevante, comentarii care au nevoie de explicații și documentarea API-ului care nu este public.

În interiorul claselor este recomandată o anumită formatare a codului, astfel încât sa fie cât mai ușor citibil. Ordinea recomandată este scrierea constantelor, urmate de declarațiile variabilelor, urmate de metode care la rândul lor vor fi așezate în funție de privilegiile oferite. metodele publice vor fi așezate înaintea celor private cu excepția în care metoda privată este folosită o singură dată și atunci este scrisă după metoda publică pe care o apelează.

Așadar s-au descris mai sus câteva principii care au fost considerate importante în implementarea proiectului. Robert C. Martin [38] în cartea Clean Code scrie în detaliu și cu exemple despre acestea și multe altele: Formatare, Obiecte și structuri de date, tratarea erorilor, limite, teste, etc.

4.5. Cazuri de utilizare

Cazurile de utilizare sunt rezultatul unei analize detaliate a cerințelor și oferă o descriere generală a modului de utilizare a sistemului. În continuare vor fi prezentați actorii sistemului și cazurile de utilizare pentru fiecare actor și acțiune posibilă în parte.

4.5.1. Actorii sistemului

Pentru interacțiunea cu sistemul s-au creat mai multe tipuri de utilizatori. Aceștia pot să interacționeze cu sistemul mai mult sau mai puțin în funcție de drepturile primite. În continuare vor fi descrise în mai multe subpuncte atât tipurile de utilizatori cât și drepturile și funcționalitățile ficăruia.

4.5.1.1. Jucător neautentificat

Jucătorul neautentificat este acel tip de utilizator care dorește accesarea funcționalităților de bază (Figura 4.10), dar preferă în același timp să rămână anonim față de sistem. Acest mod permite jucătorilor să acceseze majoritatea funcționalităților, dar spre deosebire de jucătorii autentificați, ei nu vor putea alege camera în care să joace. Asfel serverul va alege aleator camera în care playerul va fi poziționat, șansele fiind reduse de a juca împreună cu alți amici.

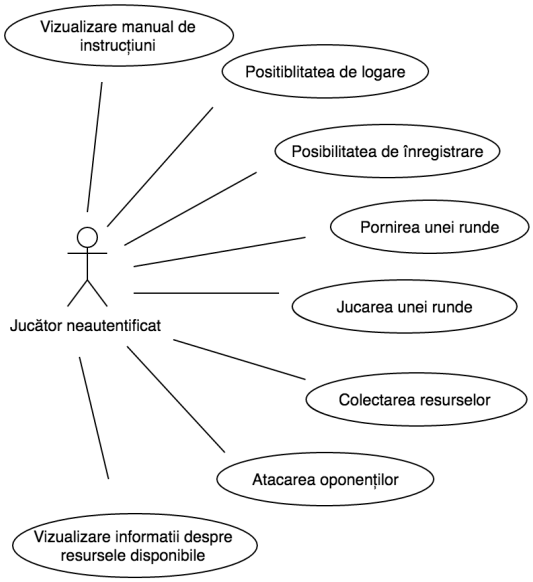


Figura 4.10 Cazuri de utilizare ai unui utilizator neautentificat

4.5.1.2. Jucător autentificat

Jucătorul autentificat este acel tip de utilizator care dorește a primi mai multe drepturi. Pentru autentificare utilizatorului îi va fi cerut să se înregistreze și să se autentifice cu contul de facebook. Sistemul va crea astfel un utilizator în baza de date și îi va asigna drepturi de jucător autentificat. Pe lângă funcționalitățile de bază care sunt prezente și în cazul unui jucător neautentificat, acesta va primi și drepturi pentru a alege camera de joc în care să participe. Astfel fiind posibil jocul împreună cu prietenii.

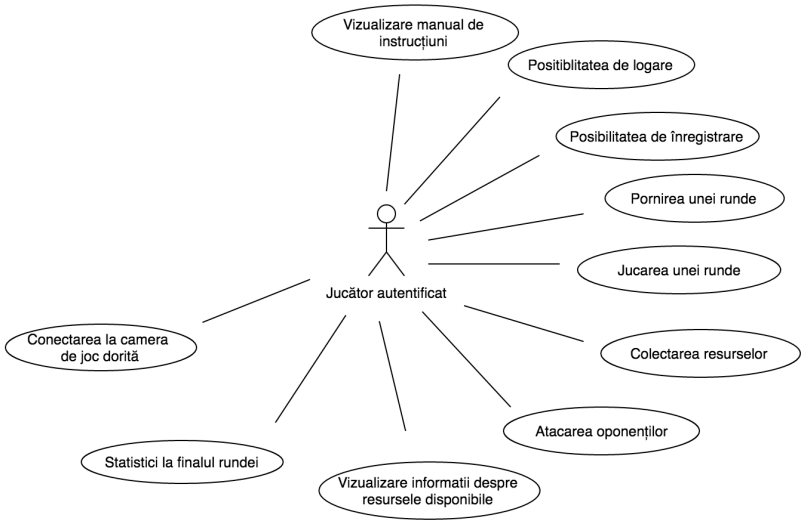


Figura 4.11 Cazuri de utilizare ai unui utilizator autentificat

4.5.2. Cazuri de utilizare pentru jucătorii autentificaţi şi neautentificaţi

În continuare vor fi prezentate cazurile de utilizare pentru utilizatorii sistemului. Jucătorii autentificați și cei neautentificați reprezintă principalii utilizatori cărora le este adresată aplicația.

4.5.2.1. Cazul de utilizare 1

Numele cazului de utilizare: Începerea unei runde ca și jucător neautentificat
Actor principal: Jucător neautentificat
Precondiții: Jucătorul se află pe pagina de start a jocului
Postcondiții: Runda a început, jucătorul trebuie să se joace

Scenariul de succes:

1. Utilizatorul se află pe pagina de start a jocului
2. Utilizatorul își alege un nume cu care va fi identificat pe parcursul unei runde
3. Utilizatorul apasă butonul de start
4. Sistemul redirecționează jucătorul către pagina de joc unde o rundă se află în desfășurare.

Scenarii alternative:

1. (alternativa punctului 1) Utilizatorul nu este pe pagina de start a jocului.
2. (alternativa punctului 2) Utilizatorul introduce caractere nepermise, ceea ce nu ii va permite conectarea la o cameră de joc.
3. (alternativa punctului 3) Utilizatorul apasă un alt buton, iar sistemul îl va redirecta spre o altă pagină decât cea dorită.
4. (alternativa punctului 4) Datorită unei probleme tehnice, sistemul nu poate redirecționa jucătorul spre camera de joc.

4.5.2.2. Cazul de utilizare 2

Numele cazului de utilizare: Începerea unei runde ca și jucător autentificat
Actor principal: Jucător autentificat
Precondiții: Jucătorul este pe pagina de start a jocului
Postcondiții: Runda a început, jucătorul trebuie să se joace

Scenariul de succes:

1. Utilizatorul se află pe pagina de start a jocului
2. Utilizatorul apasă butonul de connectare prin metoda de autentificare cu Facebook.
3. Utilizatorul își alege un nume cu care dorește să fie identificat pe parcursul rundei de joc.
4. Utilizatorul apasă butonul de start.
5. Sistemul redirecționează jucătorul către pagina de joc unde o rundă se află în desfășurare.

Scenarii alternative:

1. (alternativa punctului 1) Utilizatorul nu este pe pagina de start a jocului.

2. a. Utilizatorul nu are cont pe facebook, iar sistemul nu ii va permite autentificarea.

2. (alternativa punctului 2) Serverele de la facebook nu răspund, iar autentificarea este imposibilă.

3. (alternativa punctului 2) Utilizatorul introduce caractere nepermise, ceea ce nu ii va permite conectarea la o cameră de joc.

4. (alternativa punctului 4) Utilizatorul apasă un alt buton, iar sistemul îl va redirecta spre o altă pagină decât cea dorită.

5. (alternativa punctului 5) Datorită unei probleme tehnice, sistemul nu poate redirecționa jucătorul spre camera de joc.

4.5.2.3. Cazul de utilizare 3

Numele cazului de utilizare: Jucarea unei runde
Actor principal: Jucător autentificat sau neautentificat
Precondiții: Jucătorul a apăsat butonul de start
Postcondiții: Runda s-a terminat sau jucătorul a fost înfrânt

Scenariul de succes:

1. Sistemul direcționează jucătorul spre o cameră de joc.
2. Jucătorul este poziționat într-un loc aleator pe hartă și este afișat sub forma unui cerc în centrul ecranului.
3. Jucătorul începe să se deplaseze folosind săgețile de la tastatură, el rămânând centrat în mijlocul ecranului, dar restul obiectelor vor fi translatate în partea opusă mișcării.
4. Jucătorul va colecta resurse deplasându-se peste ele
5. La întâlnirea cu un oponent, jucătorul are opțiunea de a îl ataca.
6. Se vor repeta pașii 3-6 până în momentul în care serverul decide dacă runda s-a terminat sau jucătorul a fost învins.
7. Se vor afișa statistici relevante jucătorului în legătură cu acțiunile desfășurate de-a lungul partidei de joc.

Scenarii alternative:

1. (alternativa punctului 1) Sistemul nu a putut redirecționa jucătorul spre o cameră de joc din cauza unei erori de sistem.
2. (alternativa punctului 3) Jucătorul este atacat de oponenți, iar acesta este înfrânt, așadar este deconectat din camera de joc.
3. (alternativa punctului 6) Utilizatorul părăsește camera de joc înainte de finalul jocului, fiind deconectat automat de la camera de joc, ne mai primind statistici despre partida de joc.

4.5.2.4. Cazul de utilizare 4

Numele cazului de utilizare: Afișare instrucțiuni
Actor principal: Jucător autentificat sau neautentificat
Precondiții: Jucătorul se află pe pagina de start
Postcondiții: Instrucțiunile au fost afișate, utilizatorul a fost informat despre cum va decurge jocul.

Scenariul de succes:

1. Utilizatorul intră pe pagina de start.
2. Utilizatorul apasă butonul de instrucțiuni afișat in colțul din dreapta jos.
3. Sistemul va afișa un popup cu instrucțiunile jocului.
4. Utilizatorul va citi instrucțiunile.
5. Utilizatorul va închide popup-ul.

Scenarii alternative:

1. (alternativa punctului 2) Utilizatorul nu găsește butonul de instrucțiuni din cauza unei defecțiuni neprevăzute.
2. (alternativa punctului 3) Popup-ul nu se deschide, sau browser-ul îl blochează.

4.5.2.5. Cazul de utilizare 5

Numele cazului de utilizare: Înregistrarea unui utilizator nou
Actor principal: Jucător neautentificat
Precondiții: Jucătorul nu este autentificat și se află pe pagina de start
Postcondiții: Jucătorul este autentificat și primește mai multe privilegii

Scenariul de succes:

1. Utilizatorul se află pe pagina de start a jocului.
2. Utilizatorul apasă butonul de connectare prin metoda de autentificare cu Facebook.
3. Sistemul deschide un popup care îi cere utilizatorului să confirme cererea de conectare prin facebook.
4. Utilizatorul apasă butonul de accept.
5. Sistemul redirecționează jucătorul către pagina de start și îi adresează un mesaj de bun venit.

Scenarii alternative:

1. (alternativa punctului 2) Butonul de logare prin Facebook este inactiv datorită unei probleme de javascript.
2. (alternativa punctului 2) Utilizatorul nu dorește să se înregistreze sau să acorde informații considerate confidențiale, astfel singura modalitate de joc rămâne ca jucător neautentificat.
3. (alternativa punctului 2) Serviciile oferite de Facebook sunt inactive, utilizatorul va fi nevoit să se joace ca și jucător neautentificat.

4. (alternativa punctului 4) Utilizatorul nu este logat pe Facebook și i se cere logarea pentru a putea continua.

5. (alternativa punctului 4) Utilizatorul nu are cont pe Facebook și i se cere să se înregistreze, urmând pașii stabiliți de Facebook.

4.5.2.6. Cazul de utilizare 6

Numele cazului de utilizare: Autentificarea jucătorului în sistem
Actor principal: Jucător neautentificat
Precondiții: Jucătorul nu este autentificat și se află pe pagina de start
Postcondiții: Jucătorul este autentificat și primește mai multe privilegii

Scenariul de succes:

1. Utilizatorul se află pe pagina de start a jocului.
2. Utilizatorul apasă butonul de connectare prin metoda de autentificare cu Facebook.
3. Sistemul îi adresează un mesaj de bun venit.

Scenarii alternative:

1. (alternativa punctului 2) Butonul de logare prin Facebook este inactiv datorită unei probleme de javascript.
2. (alternativa punctului 2) Utilizatorul nu este logat pe Facebook și i se cere logarea pentru a putea continua.
3. (alternativa punctului 2) Utilizatorul nu a fost înregistrat înainte, iar sistemul îl va redirecționa automat spre pașii de înregistrare.
4. (alternativa punctului 2) Serviciile oferite de Facebook sunt inactive, utilizatorul va fi nevoit să se joace ca și jucător neautentificat.

4.5.2.7. Cazul de utilizare 7

Numele cazului de utilizare: Acesarea unei pagini restricționate către o anumită categorie de jucători
Actor principal: Jucător neautentificat sau autentificat
Precondiții: Jucătorul se află pe o pagină care nu este restricționată
Postcondiții: Pagina de acces este restricționată

Scenariul de succes:

1. Jucătorul încearcă să acceseze o pagină pe care nu are suficiente privilegii.
2. Sistemul redirecționează automat utilizatorul spre pagina de start.

4.5.2.8. Cazul de utilizare 8

Numele cazului de utilizare: Colectarea resurselor
Actor principal: Jucător neautentificat sau autentificat
Precondiții: Jucătorul se află în timpul rundei de joc
Postcondiții: Creșterea în masă a cercului controlat de jucător

Scenariul de succes:

1. Jucătorul se deplasează pe hartă și observă o resursă.
2. Jucătorul se va deplasa la resursă.
3. În momentul în care resursa se află în contact cu raza cercului controlat de jucător, sistemul va elimina resursa de pe harta de joc.
4. Cercul controlat de jucător va crește în masă.

Scenarii alternative:

1. (alternativa punctului 1) Jucătorul nu găsește resurse pe hartă datorită unei erori în sistem.
2. (alternativa punctului 3) Raza cercului nu intră în contact cu resursa, iar aceasta va rămâne în aceeași locație, masa cercului jucătorului nu va fi afectată.

4.5.2.9. Cazul de utilizare 9

Numele cazului de utilizare: Atacarea inamicilor
Actor principal: Jucător neautentificat sau autentificat
Precondiții: Jucătorul se află în timpul rundei de joc
Postcondiții: Inamicul este distrus

Scenariul de succes:

1. Jucătorul se deplasează pe hartă și observă un oponent.
2. Jucătorul va îndrepta mouse-ul spre oponentul său.
3. În momentul în care jucătorul este sigur că direcția mouse-ului este direcționată spre oponent, jucătorul va apăsa clic stânga pentru lansarea unei mingi care se va îndrepta spre inamic.
4. Inamicul este lovit, masa inamicului va scădea drastic.
5. Se vor repeta pașii anteriori până când inamicul va fi înfrânt și deconectat de la camera de joc.

Scenarii alternative:

1. (alternativa punctului 1) Jucătorul este atacat de oponent și este înfrânt
2. (alternativa punctului 3) Jucătorul are masa prea mică pentru a putea arunca mingi de atac

Capitolul 5. Proiectare de detaliu și implementare

În acest capitol se va descrie proiectarea sistemului care a fost conceput ca și aplicație client - server. Pentru partea de client se vor intra atât în detalii legate de implementarea algoritmilor pentru afișarea componentelor pe pagină în decursul jocului cât și metode care au fost folosite pentru comunicarea cu serverul. Pentru partea de server se va detalia comunicarea cu clientul și algoritmii folosiți pentru diferite metode de optimizare, astfel încât timpul de întârziere al răspunsului spre client să fie cât mai mic.

5.1. Arhitectura sistemului

S-a optat pentru o aplicație web separată în două componete mai mari: client și server. Pentru partea de client, denumită și frontend, s-a implementat interfața grafică la care vor avea acces clienții. La partea de server, denumită și backend, a fost implementată o mare parte din logica aplicației, fiind de asemenea și partea care servește informații în paralel tuturor clienților cu informații despre starea curentă a sistemului și a camerelor de joc. În Figura 5.1 se arată arhitectura conceptuală a sistemului și metodele de comunicare dintre frontend și backend.

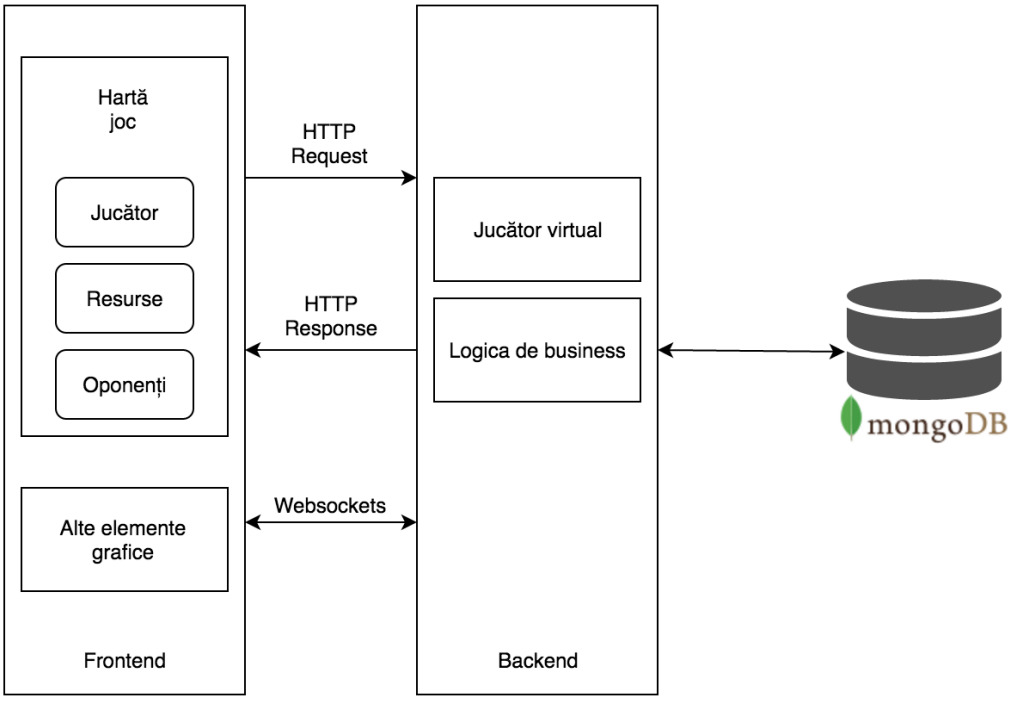


Figura 5.1. Arhitectura conceptuală a sistemului

În continuare va fi prezentată diagrama de componente a sistemului. Diagrama va cuprinde fiecare componentă care este prezentă în sistem atât pentru partea de interfață grafică cât și pentru partea de server.

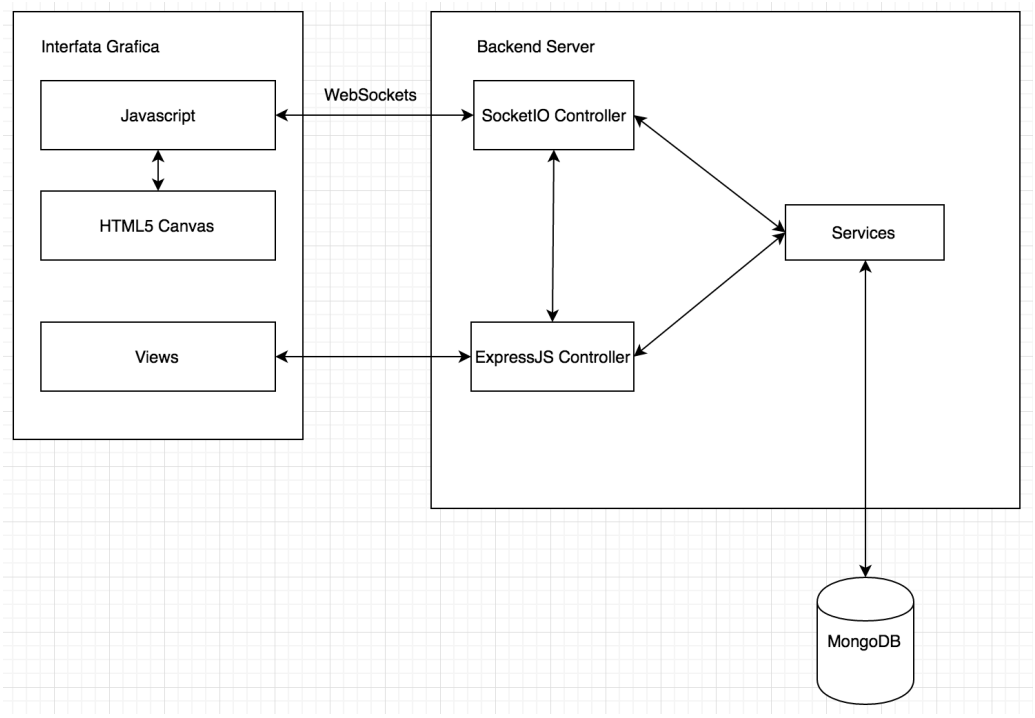


Figura 5.2 Diagrama de componente a sistemului

5.2. Elemente de implementare

5.2.1. Arhitectura componentei de interfață grafică

Interfața grafică reprezintă componenta vizuală prezentă între utilizator și partea de backend unde se face procesarea datelor care sunt servite pentru toți utilizatorii conectați la un canal aferent socket-ului la care sunt connectați. Pentru interfața grafică am ales metode standard de implementare și librării bine cunoscute în comunitățiile programatorilor de javascript de pe internet.

Interfața grafică este împărțită în două layere:

- Layer de conținut static - reprezintă view-urile generate și servite de către backend care prezintă utilizatorului pagina de primire, instrucțiuni, pagina de logare, etc.

- Layer de conținut actualizat dinamic în timp real - reprezintă un canvas încărcat cu obiecte bidimensionale sincronizate cu datele de pe backend care afișează conținutul propriuzis al jocului (jucătorul, oponenții, resursele).

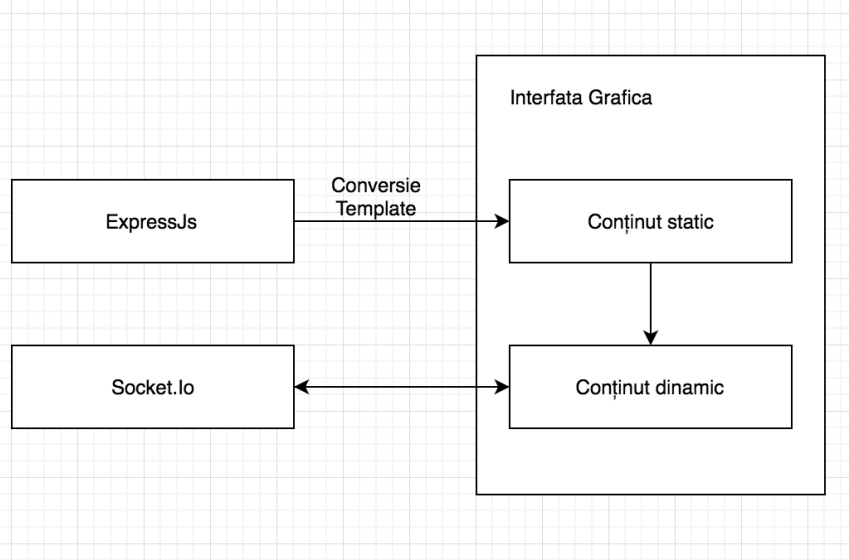


Figura 5.3 Arhitectura componentei de interfață grafică

5.2.1.1. Layer de conținut static

Layer-ul de conținut static este format din componente, care nu au continut dinamic, generate de backend pe baza unor template-uri predefinite și a unor scripturi necesare interacțiunii utilizatorului cu sistemul. Acest layer include pagina de start și pagini adiacente care împreună duc la startul jocului propriu zis.

Conținutul acestor pagini a fost scris intr-un limbaj similar HTML, numit Jade, care este compilat la cerere de către compilatorul propriu și este servit clientului in format HTML/Javascript. Unul din avantajele folosirii acestui template engine este posibilitatea de a introduce cod Javascript generat dinamic, customizat special pentru fiecare utilizator. Pe lângă acest beneficiu, mai este posibilă și încarcarea fișierelor de configurație care poate coincide cu cel al serverului de backend, ne fiind nevoie să avem configurații duplicate pentru anumite componente care sunt comune.

Acest layer face legătura, și tot odată pregătește componentele necesare funționării layer-ului de conținut dinamic. Pregătirea librăriilor necesare, al canvas-ului și al codului Javascript executat pe layerul următor este strict necesară astfel facându-se posibilă comunicarea dinamică cu serverul de backend.

Pagina de start este responsabilă cu colectarea datelor de la utilizator, astfel în momentul apăsării butonului de start, aceste date vor fi trimise către server, așteptându-se pentru o confirmare din partea acestuia cu informațiile despre joc și resursele necesare.

Exemplul următor arată codul Jade înainte și după compilare, astfel făcându-se o comparație între cele două. Se poate observa cât de ușor este de citit și de înțeles codul scris în Jade și totodată cu o complexitate mai mare decât al codului html (dat fiind faptul că în HTML nu se poat importa conținut generat dinamic).

```
doctype html
html
    head
        title FLIB
        link(rel='stylesheet', href='resources/stylesheets/style.css')
    body
        #debug
        #loginWrapper
            label(for='playerNameInput') Username
            input#playerNameInput(type='text', placeholder='Guest')
            br
            button#startButton(type='button') Play
        canvas#game
        script.
            var global = !{JSON.stringify(expose.global)}
        script(src='/socket.io/socket.io.js')
        script(src='resources/javascripts/create.js')
        script(src='resources/javascripts/player.js')
        script(src='resources/javascripts/game.js')
        script(src='resources/javascripts/app.js')
```

```
<!DOCTYPE html>
<html>
   <head>
      <title>FLIB</title>
      <link rel="stylesheet" href="resources/stylesheets/style.css">
   </head>
   <body>
      <div id="debug"></div>
      <div id="loginWrapper">
      <label for="playerNameInput">Username</label>
      <input id="playerNameInput" type="text" placeholder="Guest"><br>
      <button id="startButton" type="button">Play</button></div>
      <canvas id="game"></canvas>

      <script>var global =
      {
         "port":3000,
         "mapRadius":5000,
         "defaultPlayerRadius":30,
         "networkUpdateFactor":60,
         "maxHeartbeatInterval":5000,
         "maxResources":5000,
         "minResourcesOfType":1000,
         "initialMass":20,
         "initialRadius":20,
         "growingFactor":0.01,
         "seenDistance":2000
      }
      </script>

      <script src="/socket.io/socket.io.js"></script>
      <script src="resources/javascripts/create.js"></script>
      <script src="resources/javascripts/player.js"></script>
      <script src="resources/javascripts/game.js"></script>
      <script src="resources/javascripts/app.js"></script>
   </body>
</html>
```

Figura 5.4. Transformarea codului Jade în cod HTML

5.2.1.2. Layer de conținut dinamic

Layer-ul de conținut dinamic este un strat așezat peste layer-ul de conținut static care menține o comunicare continuă cu partea de backend, trimițându-se actualizări de la client către server și invers aproape în timp real. Frecvența de actualizare a datelor pe partea de client se face la o viteză de șaizeci de cadre pe secunde. Comunicarea dintre client și server se face aici prin websockets cu ajutorul librăriei Socket.IO. Cu ajutorul acestei tehnologii a fost posibilă implementarea unei componente grafice care să afișeze întreaga listă cu jucătorii dintr-o anumită cameră de joc ca și obiecte 2D, de asemenea și toate resursele și proiectilele care aparțin de joc. Toate acestea sunt actualizate în timp real de jucătorii conectați, iar informațiile sunt identice la toți jucătorii.

În continuare voi prezenta elementele care fac parte din acest layer.

1. Componenta de websockets - implementare cu ajutorul socket.io

Aceasta componentă are rolul de a comunica în timp real cu serverul și de a actualiza datele stocate local în browserul utilizatorului cu datele primite de la server care la rândul său preia informații de la toți utilizatorii conectați, actualizandu-și informațiile stocate în memorie pe baza acțiunilor făcute de aceștia

Conectarea la websocket se face pe baza adresei IP a serverului unde există un port deschis special pentru conexiuni de acest tip.

```
var setupSocket = function() {
      socket = io(ipAddr);
}
```

Figura 5.5 Exemplu de conexiune la websocket

Fiind un server hostat la aceeași adresa ca a serverului de ExpressJs care servește conținutul static, adresa ip nu este strict necesară, această variabilă putând avea valoarea null. În acest caz librăria socket.io se va conecta automat la serverul care a servit pagina curentă.

După asamblarea conexiunii urmează un schimb de informații care permite atât serverului cât și clientului web să își configureze conexiunea. Acest schimb de informații v-a ajuta serverul sa identifice utilizatorul conectat cu un nume de utilizator, determinat dinainte de acesta, iar pentru partea de client, browserul va primi informații referitoare la camera la care trebuie utilizatorul să se conecteze.

Următorul pas în implementare a fost crearea unor canale pe care se pot transmite date (definite pe partea de server) și a altor canale pe care se vor primi informații, astfel creându-se o mapare a datelor care vor circula prin websocket. Această mapare are datoria de a transmite anumite obiecte primite de la server ca și parametri ai unor funcții care vor manipula mai departe informația.

```
var setupSocket = function() {
      socket = io();

      socket.on('room', function(info) {
            socket = io('/' + info.room);
       socket.emit('hello', info.playerName);

       socket.on('beat', function() {
           global.latency = Date.now() - global.timeNow;
           global.beatsSkipped = 0;
       });

       socket.on('welcome', function(player) {
           global.player = new Player(player);

           createjs.Ticker.setFPS(60);
           createjs.Ticker.addEventListener("tick", handleTick);

           setupMouseListeners();
       });

       socket.on('updatePlayersMoves', function(p) {
           game.players = p;
       });

       socket.on('updateResources', function(r) {
           game.resources = r;
       });

       socket.on('updatePlayerMass', function(m) {
           global.player.updateMass(m);
       });

       socket.on('updateBullets', function(b) {
           game.bullets = b;
       });

       socket.on('dead', function() {
           handlePlayerDead();
       });
   });
};
```

Figura 5.6 Conectarea la canalele pe care se transmit date

2. **Componenta de interfață grafică - implementare cu ajutorul EaselJs**

Această componentă are ca scop afișarea informațiilor colectate de la server. O dată cu introducerea HTML5, am asistat la introducerea unor elemente menite să îmbunătățească experiența utilizatorilor cu browserul. Astfel a fost introdus și elementul Canvas, care oferă posibilitatea de rendering programatic a modelelor 2D și a imaginilor de tip bitmap.

Cu ajutorul librăriei EaselJS, care oferă soluții pentru interacțiunea cu HTML5 Canvas, am implementat câteva funcții care ajută la afișarea jucătorilor, a resurselor și a proiectilelor pe harta de joc. De asemenea background-ul este creat dinamic pe baza mișcărilor făcute de jucător pentru o mai bună experiență vizuală.

În EaselJS fiecare element vizual are la bază o instanță a clasei Shape. De asemenea fiecare din componentele acestui proiect folosesc această clasă de bază pentru o manipulare mai ușoară a acestora.

2.1. Player

Fiecare jucător este incapsulat intr-un obiect de tip Player, care conține cinci proprietăți folosite în joc, atât pentru identificare și poziționare pe harta de joc cât și anumite caracteristici vizuale.

- id - un identificator unic
- name - numele jucătorului
- resources - numărul de resurse colectate de jucător
- mass - greutatea jucătorului în funție de numărul de resurse colectate
- shape - obiect de tip Shape, specific easel.js pentru randarea jucătorului

```
class Player {
    constructor(params) {
        this.id = params.id;
        this.name = params.name;
        this.resources = params.resources;
        this.mass = params.mass;
        this.shape = this.createShape(params);
    }

    createShape(params) {
        var shape = new createjs.Shape();

        shape.graphics.setStrokeStyle(2);
        shape.graphics.beginStroke("#000");
        shape.graphics.beginFill(params.color).drawCircle(0, 0,
global.initialRadius + params.mass * global.growingFactor);
        shape.x = params.x;
        shape.y = params.y;

        return shape;
    }

    updateMass(m) {
        this.mass = m;
        this.updateShapeRadius();
    }

    updateShapeRadius() {
        this.shape.graphics.command.radius = global.initialRadius
+ this.mass * global.growingFactor;
    }
}
```

Figura 5.7. Proprietățile și metodele clasei Player

Mișcarea utilizatorului pe hartă se face prin incrementarea sau decrementarea, la fiecare tic de ceas, cu o anumită valoare, a punctelor carteziene, în funcție de tastele apăsate. Pentru aceasta s-a facut key binding pentru tastele W - sus, A - stânga, S - jod, D - dreapta.

```
document.onkeydown = keydown;
document.onkeyup = keyup;

function keydown(event) {
	keys[event.keyCode] = true;
}

function keyup(event) {
	delete keys[event.keyCode];
}


static handlePlayerInput(keys) {
	if (keys[65]) global.player.shape.x -= 3;
	if (keys[87]) global.player.shape.y -= 3;
	if (keys[68]) global.player.shape.x += 3;
	if (keys[83]) global.player.shape.y += 3;
}

var handleTick = function() {
	Game.handlePlayerInput(keys);

	var newPosition = {x: global.player.shape.x, y:
global.player.shape.y};
	socket.emit("playerMove", newPosition);

	 ...
};
```

Figura 5.8. Procesarea informațiilor de la tastatură și maus

În cazul în care jucătorul este distrus de proiectilele oponenților, conexiunea la socket v-a fi intrerupta și utilizatorul v-a fi redirectionat la pagina de start.

```
var handlePlayerDead = function() {
	document.getElementById("loginWrapper").style.display =
"block";
	game = new Game({"canvasId": "game"});
	socket.disconnect();
};
```

Figura 5.9 Manipularea conexiunii în cazul unui jucător distrus în timpul jocului

2.2. Resource

Resursele vizibile pe harta de joc sunt generate complet pe partea de backend, iar la fiecare update primit prin websocket se vor afișa doar cele incluse în reginuea de interes a utilizatorului. S-a preferat formarea obiectului de resurse pe partea de backend, iar pentru partea de frontend, singura responsabilitate rămânand extinderea clasei Shape și afișarea acestora.

```
handleGraphics() {
    ...
    //draw resources
    this.resources.blue.forEach(function(r) {
        self.drawResource(r);
    });

    this.resources.yellow.forEach(function(r) {
        self.drawResource(r);
    });

    this.resources.red.forEach(function(r) {
        self.drawResource(r);
    });
    ...
}

drawResource(resource) {
        var resourceShape = new createjs.Shape();
        resourceShape.graphics.beginFill(resource.color).drawCircle(0, 0,
resource.mass);
        resourceShape.x = resource.x;
        resourceShape.y = resource.y;
        this.stage.addChild(resourceShape);
}
```

Figura 5.10. Manipularea resurselor pe interfața grafică

2.3. Bullets (proiectile)

Proiectilele vizibile pe harta de joc sunt de asemenea formate și trimise de către server în funcție de aria vizuală a jucătorului. Datorită numărului mare de update-uri pe secundă, se poate observa o mișcare similară realității datorită efectului grafic realizat prin modificarea punctelor carteziene a proiectilelor.

```
drawBullet(bullet) {
        var bulletShape = new createjs.Shape();
        bulletShape.graphics.beginFill("red").drawCircle(0, 0,
bullet.radius);
        bulletShape.x = bullet.x;
        bulletShape.y = bullet.y;
        this.stage.addChild(bulletShape);
}
```

Figura 5.11. Manipularea proiectilelor pe interfața grafică

2.4. Background

Pentru o mai bună experiență vizuală s-a hotărât ca fundalul să nu fie static, ci cu ajutorul unui algoritm s-a reușit producerea unui efect care permite fundalului să se miște în sens opus mișcărilor efectuate de jucător. Algoritmul se bazează pe mișcarea liniilor generate cu ajutorul clasei Shape la o viteză echivalentă cu cea a jucătorului.

```
drawBackground() {
	// background color
	var backgroundShape = new createjs.Shape();
	backgroundShape.graphics
			.beginFill("#F0F2EB")
				.drawRect(0,0,
					this.stage.canvas.width,
					this.stage.canvas.height);
	backgroundShape.x = global.player.shape.x -
this.stage.canvas.width / 2;
	backgroundShape.y = global.player.shape.y -
this.stage.canvas.height / 2;
	this.stage.addChild(backgroundShape);

	// grid vertical lines
	var w = this.stage.canvas.width;
	var x = w / 50; // 50 vertical lines
	while (w >= 0) {
		w = w - x;
		var line = new createjs.Shape();
		line.graphics.setStrokeStyle(3);
		line.graphics.beginStroke("#E1EDB9");
		line.graphics.moveTo(global.player.shape.x -
this.stage.canvas.width / 2 + w - global.player.shape.x % x,
global.player.shape.y - this.stage.canvas.height / 2);
		line.graphics.lineTo(global.player.shape.x -
this.stage.canvas.width / 2 + w - global.player.shape.x % x,
global.player.shape.y + this.stage.canvas.height);
		line.graphics.endStroke();
		this.stage.addChild(line);
	}

	//algoritm similar pentru liniile orizontale
}
```

Figura 5.12. Cod responsabil pentru desenarea background-ului

5.2.2. Arhitectura componentei de backend

Serverul de backend-ul are un rol foarte important în funcționarea unui sistem multiplayer. Această componentă este absolut necesară pentru validarea datelor de la utilizatori și pentru a menține jocul corect față de toți jucătorii. Există însă posibilitatea implementării fără un server care să fie intermediar între jucători, iar toată validarea să fie făcută local la jucători în browser. Această opțiune este foarte periculoasă întrucât s-ar putea introduce cod malițios în request care să fie executat la toți cei conectați într-un joc sau chiar să execute mutări care să îl avantajeze într-un fel sau altul. Așadar este nevoie de o componentă intermediară care să filtreze datele care se trimit între utilizatori.

Componenta de backend folosește o arhitectura multi-tier compusă din trei layere, fiecare având un rol important și bine determinat. Aceste layere comunică între ele pentru a putea genera un răspuns înainte de a fi trimis la un utilizator.

Primul layer, unde se face comunicarea directă cu utilizatorul, este controller. Acest layer este divizat în două tipuri de controllere. Primul tip este controller-ul care coordoneaza paginile statice generate cu ajutorul ExpressJS pentru afișarea conținutului din pagina de primire, login, ajutor, etc. Al doilea tip de controller este cel care comunică în timp real cu ajutorul websocket-urilor, prin intermediul librăriei SocketIO

disponibilă și în componenta interfeței grafice.

Al doiela layer este folosit pentru partea de business a aplicației care are rolul de a filtra acțiunile jucătorilor și de a le folosi în vederea calculării mișcărilor jucătorilor și a gloanțelor folosite de aceștia în scopul distrugerii inamicilor.

Ultimul layer este folosit pentru persistență, are rolul de a face legătura dintre stratul de bussiness și baza de date.

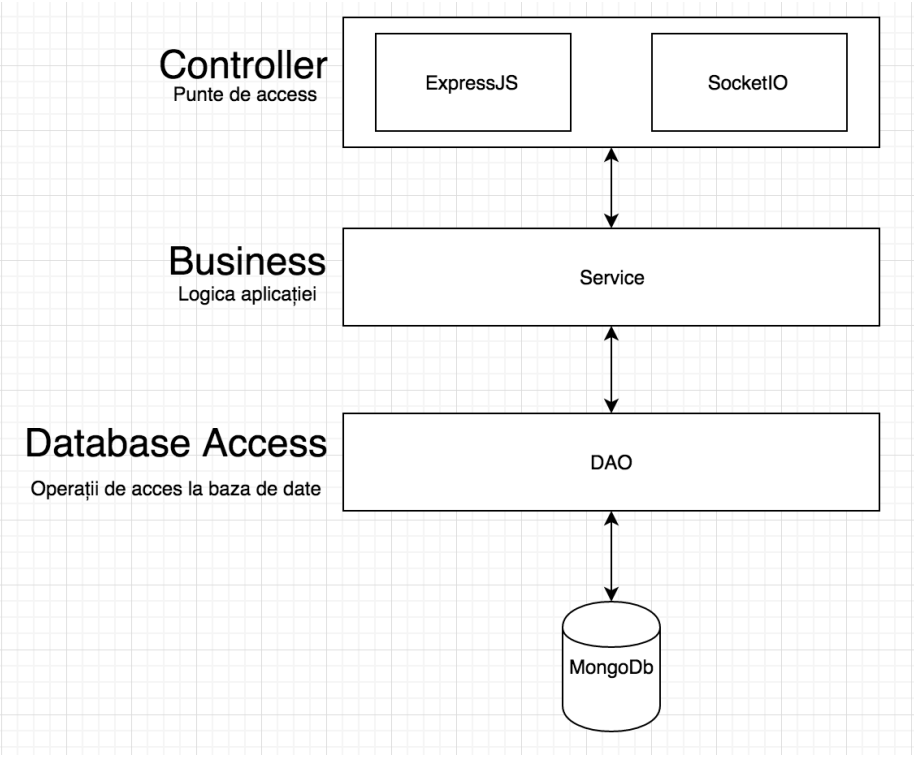


Figura 5.13. Arhitectura componentei de backend

5.2.2.1. Controller layer

Controller layer este responsabil de interceptarea mesajelor de la client spre backend si de trimiterea răspunsurilor de la backend spre client. Acest layer este la randul său subdivizat pentru creșterea performanței. Există două mari componente care comunică cu clientul:

- Componenta de conținut static - Această componentă se ocupă de requesturi pentru care nu este necesară accessarea de multe ori și care sunt gândite pentru a fi executate doar când userul accesează interfața grafică până în momentul în care jocul începe.

- Componenta de conținut dinamic servit in timp real - Această componentă se ocupă de requesturi pentru care e nevoie sa fie executate intr-un număr foarte mare în timpul jocului. Ținând cont că jocul trebuie actualizat la o rată de 60 cadre pe secundă, aceste mesaje sunt trimise la un interval de câteva milisecunde. Aceste mesaje se trimit prin intermediul unui singur request care rămâne ca și un canal deschis prin care se

trimit doar mesaje care pot fi captate de client. Aceste mesaje au rolul de a trimite informații despre locațiile jucătorilor și a resurselor de pe harta de joc.

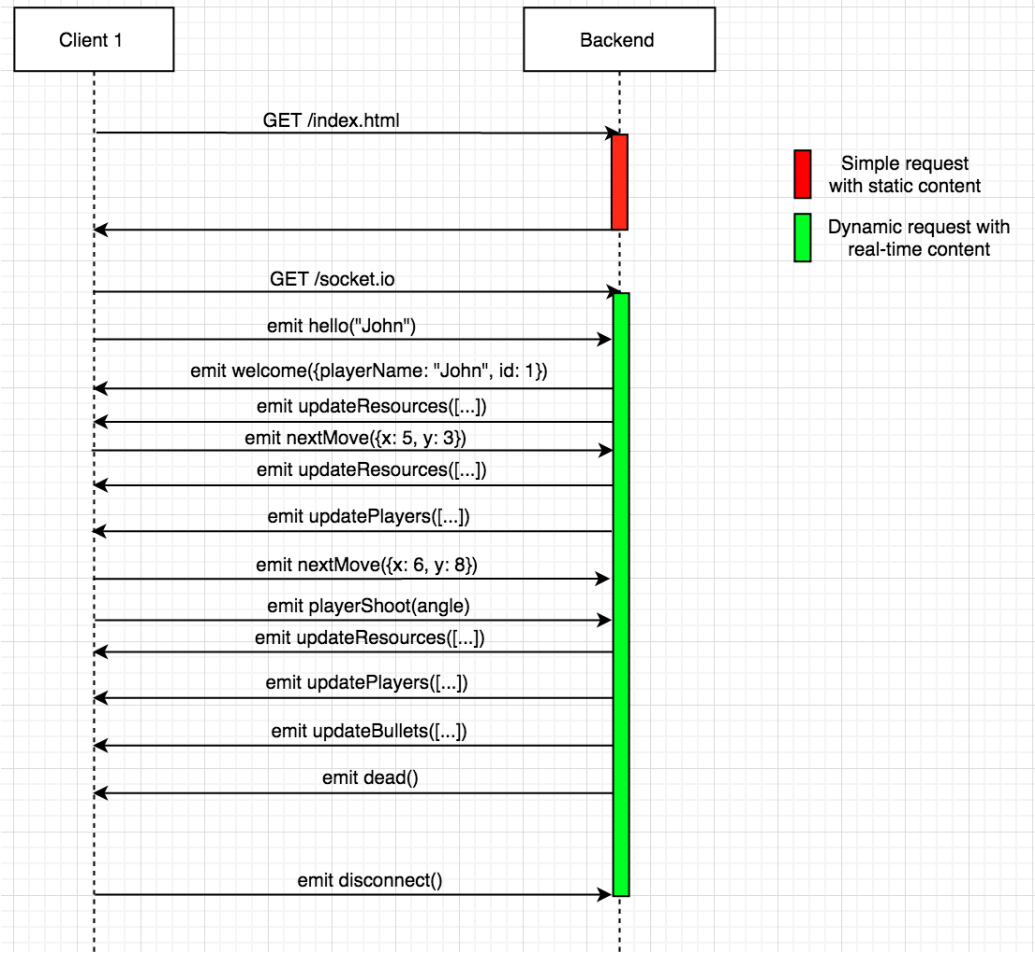


Figura 5.14 Exemplu de requesturi posibile între client și backend

5.2.2.2. Business layer

Business layer are un rol foarte important în partea logică a aplicației. În acest layer se vor primi toate datele de la layerul de controllere legate de acțiunile utilizatorilor, aici urmând a se face calculele privind diferite mișcări, resurse captivate, etc. care un utilizator le poate acționa, urmând a fi trimise înapoi spre layerul de controllere care va face legătura cu clientul, trimitandu-i update-uri periodice.

Acest layer este construit din trei servicii și trei componente ajutătoare, iar în continuare le voi detalia pe rând:

1. Serviciul de gestionare al jucătorilor (bullets.service.js)

Acest serviciu are datoria de a face posibilă crearea de noi jucători și de persistarea lor în baza de date. Pentru minimizarea numărului de apeluri spre baza de date s-a

preferat persistarea utilizatorilor și într-un array unidimensional astfel încât atunci cand este nevoie să obținem lista jucătorilor dintr-o anumita cameră de joc să putem folosii datele care sunt deja alocate în memorie.

Serviciu este format din două funcții:

- **getAll** - funcție care returnează array-ul cu jucătorii care sunt activi în joc
- **createPlayer** - funcție care crează un jucător pe baza informațiilor primite de la utilizator. Parametrii acestei funcții sunt **socketId** si **playerName**. SocketId reprezinta id-ul unic al socket-ului generat cu ajutorul librăriei socket.io , iar playerName este numele utilizatorului care a fost pasat o dată cu primul request al utilizatorului.
-

```
var createPlayer = function(socketId, playerName) {
    var player = new Player(socketId, playerName);
    players.push(player);
    return player;
};
```

Figura 5.15 Exemplu de cod pentru crearea unui player

Serviciul importă clasa Player care este folosită în instanțierea obiectelor de tip Player, astfel putând persista o structură relativ fixă a obiectului atât în baza de date cât și în memorie.

2. Serviciul de gestionare a resurselor (resources.service.js)

Gestionarea resurselor dintr-o cameră de joc a necesitat creearea unui serviciu care să îndeplinească, la fel ca în cazul serviciului de gestionare al jucătorilor, persistarea resurselor în memorie.

Spre deosebire de serviciul prezentat anterior, stocarea resurselor în baza de date nu este necesară întrucât sunt stocate în memorie într-un număr limitat, iar o posibilă pierdere a acestor date legate de resurse nu ar influența în mod negativ pe termen lung. Resursele sunt create automat de sistem la coordonate carteziene generate pseudo-random cu ajutorul componentei ajutătoare pentru generarea de locații.

O altă deosebire este modul de stocare al datelor. În cazul stocării datelor despre utilizatori a fost suficientă stocarea lor într-un array, pentru că numărul de jucători dintr-o cameră de joc este limitat la un număr relativ mic, iar optimizările în acest sens nu erau necesare datorită faptului că un eventual câștig de timp ar fi fost insesizabil. În cazul resurselor, care sunt într-un număr mare, iterarea asupra tuturor ar fi mărit foarte mult timpul de execuție. O iterație ar fi însemnat ”căutarea tuturor resurselor la o distanta x față de coordonatele jucătorului”. Pentru fiecare jucător era necesară o astfel de iterație astfel nivelul de complexitate ar fi fost O(n*m). Pentru a optimiza acest proces, am optat pentru folosirea librăriei SimpleQuadtree care oferă o structură de date de tip quadtree și ajută la optimizarea complexității, reducând astfel foarte mult timpul de căutare. Astfel, doar în cel mai nefavorabil caz s-ar ajunge la o complexitate asemănătoare.

Acest serviciu pune la dispoziție șapte funcții accesibile din layer-ul de controllere:

- **balanceResources** - funcție responsabilă de echilibrarea resurselor disponibile în cadrul unei camere de joc, necesară datorită fluctuațiilor aleatoare urmate din colectarea resurselor de către utilizatori.

```
var balanceResources = function () {
    var blueResources = resources.blueCount;
    var blueDeficit = config.minResourcesOfType - blueResources;

    while (blueDeficit > 0) {
        addResource("blue");
        resources.blueCount += 1;
        blueDeficit--;
    }

    // similar pentru culorile galben și roșu
};
```

Figura 5.16 Exemplu de cod pentru balansarea resurselor

- **addResource** - funcție responsabilă pentru itroducerea unei resurse în quadtree
- **decreaseBlue**, **decreaseYellow**, **decreaseRed**, cele trei funții sunt necesare pentru a face decrease la un count specific fiecarei tip de resurse. Balansarea resurselor face automat increase la acest count, asadar nu a fost nevoie de functii pentru increase.
- **remove** - funcție responsabilă de ștergerea resurselor din quadtree
- **getSeenResourcesByPlayer** - funcție responsabilă de extragerea din quadtree a obiectelor din raza vizibilă a jucătorului

```
var getSeenResourcesByPlayer = function(p) {
    var playerView = {
        x: p.x - config.seenDistance / 2,
        y: p.y - config.seenDistance / 2,
        w: config.seenDistance,
        h: config.seenDistance
    };

  return resources.get(playerView);
};
```

Figura 5.17 Exemplu de query pentru extragerea obiectelor din quadtree

3. Serviciul de gestionare al muniției (bullets.service.js)

Un alt serviciu necesar, cu o implementare asemănătoare celor prezentate anterior, a fost introdus în scopul managementului de proiectile (gloanțe) folosite în cadrul jocului de către utilizator. Întrucât numărul lor poate crește foarte mult în funcție de acțiunile jucătorilor, iar poziția unui proiectil se modifică la fiecare tic de ceas, dar în același timp trebuie verificată și distanța de la un proiectil la un jucător, a fost nevoie de o structură a datelor mai specială față de cele prezentate anterior. Soluția pentru optimizare în acest sens a fost implementarea a două structuri de date care să conțină aceleași referințe ale obiectelor (proiectilelor) de două ori:

- **array** - pentru a modifica poziția elementelor la fiecare tic de ceas s-a conclus că este necesară o parcurgere a întregii liste de, iar cunoscând unghiul la care a fost proiectat glonțul, se poate calcula poziția următoare
- **quadtree** - oferă o căutare mai rapidă a proiectilelor din jurul unui utilizator

Acest serviciu pune la dispoziție patru funcții accesibile din layer-ul de controllere:

- **createBullet** - funție responsabilă pentru crearea unui proiectil și inserarea lui în array și quadtree. Această funcție primeste ca parametrii jucătorul care a inițiat proiectilul, locația inițială a proiectilului (coordonatele x și y sunt copiate de la jucător) și unghiul la care a fost proiectat
- **checkBulletsHitPlayers** - funcție responsabilă cu iterația peste toate proiectilele și utilizatorii în vederea calculării distanței dintre fiecare pe rând. Dacă distanța calculată este mai mică sau egală cu suma razelor al jucătorului și al proiectilului, înseamnă că a avut loc o coliziune între cele două, iar o parte din masa acumulată de jucător pe parcursul jocului este transferată către utilizatorul care a inițiat proiectilul

```
var checkBulletsHitPlayers = function() {
    var players = PlayersService.getAll();

    for (var j = players.length - 1; j >= 0; j--) {
        var p = players[j];
        for (var i = bulletReferences.length - 1; i >=0; i--) {
            var b = bulletReferences[i];

            // no suicide in this game lol
            if (p == b.player)
                continue;

            if (Geometry.checkCirclesCollision(p, b, p.getRadius() + b.radius)) {
                p.mass -= b.radius * 10;
                b.player.mass += b.radius * 10;
                b.player.hit = true;

                bullets.remove(b);
                bulletReferences.splice(i, 1);

                if (p.mass <= 30)
                    p.alive = false;
            }
        }
    }
};
```

Figura 5.18 Exemplu de iterație asupra elementelor din lista cu proiectile în scopul verificării dacă acestea intră în coliziune cu jucătorii

- **updateBulletsLocation** - această funcție este apelată la fiecare tic de ceas, locația fiecărui proiectil fiind modificată în funție de unghiul, viteza și timpul scurs din momentul lansării proiectilului

```
var updateBulletsLocation = function() {
    for (var i = bulletReferences.length - 1; i >= 0; i--) {
        var b = bulletReferences[i];
        if (b.ticks > 500) {
            bullets.remove(b);
            bulletReferences.splice(i, 1);
            continue;
        }

        b.ticks += 1;

        //moveX = Math.cos(angle) * speed * time;
        //moveY = Math.sin(angle) * speed * time;
        //time = 1 tick
        b.x += Math.cos(b.direction) * 5;
        b.y += Math.sin(b.direction) * 5;
    }
```

Figura 5.19 Exemplu de calcul a poziției următoare a proiectilelor

- **getSeenBulletsByPlayer** - fiecare utilizator poate să vadă în jurul său doar pe o anumită rază ceea ce înseamnă că este nevoie de filtrarea elementelor pe baza unei regiuni de interes. În acest scop, această funcție este responsabilă de căutarea în quadtree a elementelor apropiate de utilizator.

4. Componenta ajutătoare pentru generare de locații

Această componentă ajutătoare are o singură responsabilitate și aceea de a genera locații aleatoare pentru jucători și pentru resursele create dinamic pe harta de joc. Datorită faptului că harta de joc este limitată în formă circulară, a fost necesară implementarea unui algoritm de generare random a punctelor carteziene incluse in interiorul cercului care delimitator.

Având o singură responsabilitate, această componentă este compusă doar dintr-o singură funcție care face cele descrise mai sus:

- **randomLocation** - generator de locații aleatoare în coordonate carteziene aflate în raza cercului delimitator

```
var randomLocation = function() {
    var location = {};
    var radius = config.mapRadius;

    var theta = Math.random() * 2 * 3.14159;
    var distance = Math.sqrt(Math.random()) * radius;

    location.x = distance * Math.cos(theta); // + centerX
    location.y = distance * Math.sin(theta); // + centerY

    return location;
};
```

Figura 5.20 Exemplu de generator de locații aleatoare

5. Componenta ajutătoare pentru generare de culori

Fiind tot o componentă cu o singură responsabilitate, această componentă este responsabilă de generarea culorilor asignate jucătorilor în cadrul jocului. Pentru o experiență vizuală plăcută, s-a decis implementarea unui algoritm de generare de culori ”flat” (culori considerate plăcute ochilor care nu sunt stridente sau greu de urmărit).

Asemenea componentei anterioare, această componentă are o singură funcție:

- **randomColor** - generator de culori aleatoare plăcute vizual

```
var randomColor = function() {
    var r = (Math.round(Math.random() * 127) + 127).toString(16);
    var g = (Math.round(Math.random() * 127) + 127).toString(16);
    var b = (Math.round(Math.random() * 127) + 127).toString(16);
    return '#' + r + g + b;
};
```

Figura 5.21 Exemplu de generator de culori aleatoare

6. Componenta ajutătoare pentru funcții geometrice

Această componentă are ca responsabilitate implementarea funcțiilor matematice - geometrice. Componenta este formată momentan dintr-o singură funcție responsabilă de detecția coliziunilor dintre jucători, sau dintre un jucător și un proiectil:

- **checkCriclesCollision** - funcție care primeste ca parametrii punctele carteziene ale centrelor a doua cercuri și suma razelor lor

```
var checkCirclesCollision = function(a, b, c) {
    return c >= Math.sqrt(Math.pow(b.x - a.x, 2) + Math.pow(b.y - a.y, 2));
};
```

Figura 5.22 Exemplu de verificare a coliziunii a două cercuri

5.3. Alte componente importante

5.3.1. Manager de dependințe - NPM

Datorită faptului că proiectul depinde de mai multe librării care în timp pot suferi modificări din partea dezvoltatorilor, iar anumite update-uri ar putea fi necesare, s-a hotărât folosirea unui manager de dependințe (NPM) care să ușureze efortul depus pentru schimbarea librăriilor.

La rulare, NPM va căuta automat fișierul ”package.json” care conține descrierea proiectului și a dependințelor necesare de acesta. Dependințele considerate valide, vor fi downloadate în folderul ”node_modules”, astfel NodeJS va știi de unde să folosească dependințele cerute. Dacă se dorește încărcarea în modulul de frontend a dependințelor este nevoie să se specifice tot path-ul până la acestea.

```
{
  "name": "flib",
  "version": "0.0.1",
  "description": "Flib strategy game",
  "main": "server/server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server/server.js"
  },
  "keywords": [
    "flib"
  ],
  "author": "Sergiu Falcusan",
  "engines": {
    "node": "5.8.0"
  },
  "dependencies": {
    "express": "4.14.0",
    "jade": "1.11.0",
    "requirejs": "2.2.0",
    "simple-quadtree": "0.1.3",
    "socket.io": "1.4.6"
  }
}
```

Figura 5.23 Exemplu de JSON valid specific npm

Toate dependințele descrise în Figura 5.3X au fost explicate și folosite în implementarea proiectului. Librăriile care sunt folosite de aceste dependințe nu vor fi explicate în această lucrare, deoarece nu sunt folosite în mod direct.

3.3.2. Componente comune

Pentru a nu avea configurații (care sunt folosite în joc) diferite între componentele de backend și frontend s-a optat folosirea unui fișier de configurație care să poată fi integrat de ambele module. Acest fișier conține informații despre port-ul la care ruleaza aplicația și constante folosite în joc, ca: raza cercului unui jucător, distanța la care poate vedea, dimensiunile hărții etc. În fișier se află un obiect de tip JSON, care cu ajutorul librăriei ”require.js” poate fi încărcat la cerere și transformat în obiect Javascript.

```
{
  "port": 3000,
  "mapRadius": 5000,
  "defaultPlayerRadius": 30,
  "networkUpdateFactor": 60,
  "maxHeartbeatInterval": 5000,
  "maxResources": 5000,
  "minResourcesOfType": 1000,
  "initialMass": 20,
  "initialRadius": 20,
  "growingFactor": 0.01,
  "seenDistance": 2000
}
```

Figura 5.24 Exemplu de JSON reprezentând configurația specifică a aplicației

3.3.3. Inteligență artificială - Fatboy

O componentă separată a proiectului este o mini aplicație care este responsabilă de partea de inteligență artificială a jocului și mai exact un creator de boti care să simuleze jucători reali. Această componentă a primit numele de Fatboy, datorită faptului că boții au ca prioritate să culeagă resursele de pe hartă.

Fatboy a fost dezvoltat în limbajul de programare Javascript, ca și toate celelalte componente și folosește librăria socket.io pentru a se conecta la un canal specific unei camere de joc. După conectare folosește acest canal pentru a primi și a trimite informații referitoare la mișcările făcute.

```
// receive player info
socket.on('welcome', function (p){
    player = p;
});

// if player is dead, disconnect it's socket
socket.on('dead', function() {
    socket.disconnect();
});

// persist resources object
socket.on('updateResources', function(r) {
    if (r.blue.length > 0 && r.red.length > 0 && r.yellow.length > 0)
        resources = r.blue.concat(r.red.concat(r.yellow));
});
```

Figura 5.25 Legătura dintre componenta de Inteligenta artificială și backend

A fost implementat un algoritm de tip greedy pentru a găsi un optim local bazat pe cea mai apropiată resursă vizibilă bot-ului. Pentru calculul distanței celei mai apropiate resurse, s-a calculat distanța euclidiana de la coordonatele bot-ului până la fiecare resursă. Ultimul pas a fost extragerea celei mai apropiate resurse într-o variabilă și deplasarea până la aceasta.

```
// compute euclidean distance between two points
function euclideanDistance(x1, y1, x2, y2) {
    return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
}
// compute the angle between player and resource and
function computeAngleBetweenTwoPoints(x1, y1, x2, y2) {
    return Math.atan2(y2 - y1, x2 - x1);
}
// compute nextMove using angle and given speed
function computeNextMove() {
    // angle between player pos and closest resource
    var angle = computeAngleBetweenTwoPoints(player.x, player.y,
closestResource.x, closestResource.y);

    player.x += Math.cos(angle) * speed;
    player.y += Math.sin(angle) * speed;

    return {x: player.x, y: player.y}
}
```

Figura 5.26 Calcule matematice pentru distanțe și unghiuri

```
// control user moves and fire bullets
function applyStrategy() {
    if(dead != 1) {
        closestResource = getClosestResource();

        var nextMove = computeNextMove();
        moveToPosition(Math.ceil(nextMove.x), Math.ceil(nextMove.y));

        setTimeout(applyStrategy, FPS);
    }
}
function getClosestResource() {
    var closestResourceDist = MAX_INT;
    var closestResource = {x: 0, y: 0};

    if (resources.length > 0) {
        resources.forEach(function (r) {
            var dist = euclideanDistance(player.x, player.y, r.x, r.y);
            if (dist < closestResourceDist) {
                closestResourceDist = dist;
                closestResource = r;
            }
        });
    }
    return closestResource;
}

function moveToPosition(x, y) {
    socket.emit("playerMove", {x: x, y: y});
}

applyStrategy();
```

Figura 5.27 Aplicarea strategiei de căutare și deplasare către cea mai apropiată resursă

5.4. Procesul de scalare

Crearea unui sistem care să permită conexiunea mai multor utilizatori în diferite camere de joc face parte din cerințele nonfuncționale ale lucrării și este deosebit de important în cadrul unui joc multiplayer. În continuare se va prezenta procesul de scalare al aplicației.

DigitalOcean [42] este unul din cei mai mari provideri de infrastructură din lume care oferă programatorilor servicii de cloud pentru lansarea și scalarea produselor. Acest provider oferă metode de echilibrare a sarcinilor, astfel încât utilizatorii noi veniți vor fi redirecționați către serverele mai puțin folosite.

Pentru balansarea sarcinilor s-a decis folosirea a două tipuri de servere. Primul server este dedicat sistemului de load balancing, iar cel de-al doilea va fi o imagine a unui server de joc, care în momentul lansării va porni automat o instanță a jocului. Așadar în funcție de încărcătura serverului de joc și de numărul de utilizatori conectați se va efectua clonarea imaginii care rulează jocul. O dată clonată imaginea, aceasta va fi pornită automat și se va pune la dispoziția serverului de echilibrare.

Pentru ca utilizatorii să nu fie reconectați la un alt server chiar în timpul jocului, serverul de load-balancing va trimite, la conectarea utilizatorului, un cookie care va fi preluat si transmis în toate requesturile, cu informații despre serverul la care este conectat utilizatorul.

Capitolul 6. Testare și Validare

În acest capitol se vor prezenta modalitățile de testare folosite în dezvoltarea aplicației, rolul acestor teste fiind validarea cerințelor funcționale și nonfuncționale impuse în capitolul 4.

Principalele modalități de testare folosite au fost efectuate folosind tooluri speciale pentru capturarea informațiilor despre CPU și memoria utilizată. Aceste tooluri fac parte din browser și din IDE-ul Webstorm [39]. De asemenea o metodă de testare a fost crearea botului ”Fatboy” care are rolul de a confirma funcționarea corectă a sistemului.

6.1. Testarea performanței

Testele de performanță implică calcularea timpilor în care browserul primește informații de la server, le procesează, iar apoi le afișează. Conform cerințelor, acesta trebuie să facă posibilă comunicarea în timp real și afișarea datelor la o viteză de șaizeci cadre pe secundă.

Testele s-au efectuat pe un calculator cu sistem de operare Mac OSX, procesor dual core Intel Core i7 2.7GHz și 8GB memorie RAM. Serverul a fost rulat pe acest calculator, boții (care simulează playeri reali) rulând pe alte calculatoare aproximativ la fel de performante.

6.1.1. Testare CPU

Testele de perfomanță pentru utilizarea procesorului au fost efectuate cu ajutorul unui test de latență, care verifică timpul scurs de la inițierea unui request până la primirea răspunsului. Acest test a fost mai apoi validat cu ajutorul instrumentului Javascript Profiler [40] din cadrul browserului Google Chrome. În tabelul următor sunt prezentate rezultatele testelor de performanță executate. Testele au fost efectuate rulând în paralel simulatorul Fatboy.

Nr. utilizatori concurenți	Nr. mesaje / secundă așteptate	Timp maxim de răspuns (ms)	Cadre / secundă posibile
5	150	14	71
10	300	18	55
20	600	22	45
30	900	31	32

Tabel 6.1 - Rezultatele testării performanței CPU

Din tabelul de mai sus se poate observa că timpul mediu de răspuns crește semnificativ o dată cu numărul de playeri conectați. Rezolvarea acestei probleme este prin scalarea aplicației, iar cu ajutorul unui load balancer se vor redirecționa utilizatorii

către unul din serverele mai libere. Rezultatele de mai sus au fost obținute prin rularea pe un singur server pentru a stabili volumul potențial de procesare a cererilor.

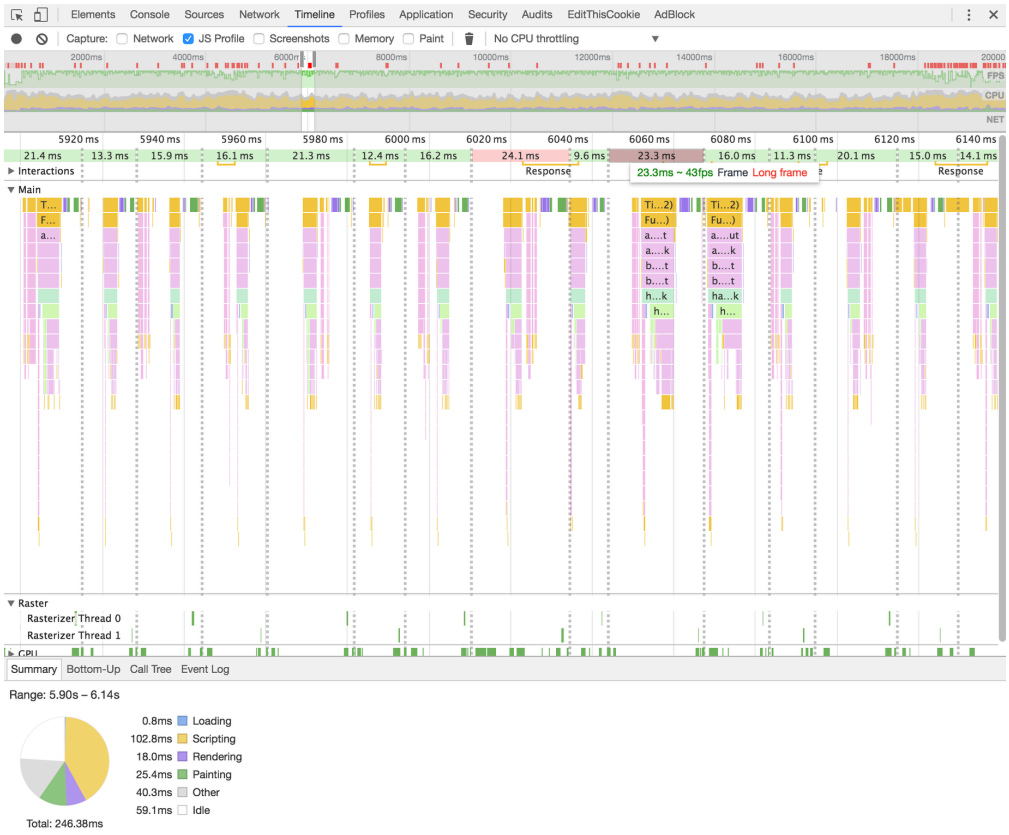


Figura 6.1 Validarea testului de performanță pe baza informațiilor din Javascript Porfiler

6.1.2. Testare memorie

Testele de performanță pentru memoria aplicației au fost efectuate cu ajutorul instrumentului ”V8 Profiling” disponisbil în IDE-ul Webstorm. Acesta oferă posibilitatea capturării întregii memorii făcând un snapshot al memoriei Heap.

Din aceste teste s-a putut observa că memoria consumată are o valoare constantă, cu mici fluctuații în funcție de acțiunile utilizatorilor. Scăpări de memorie care ar putea genera umplerea memoriei calculatorului și oprirea aplicației nu au putut fi observate.

Aceste teste au fost folositoare de-a lungul dezvoltării aplicației întrucât au existat cazuri în care memoria alocată nu se ștergea pentru că mai existau obiecte care aveau pointer înspre memoria respectivă. În aceste cazuri se putea observa că anumiți vectori, utilizați pentru gestiunea resurselor și a proiectilelor, creșteau foarte mult și astfel problema de alocare necorespunzătoare putea fi identificată ușor ulterior în cod.

6.2. Chestionar adresat utilizatorilor

Pentru a avea o idee despre primele impresii ale utilizatorilor, s-a hotărât hostarea aplicației pe un server și invitarea a zece persoane care să testeze aplicația. La sfârșit participanții au fost rugați să completeze un chestionar, generat cu ajutorul Google Forms. În continuare vor fi prezentate întrebările și răspunsurile celor chestionați.

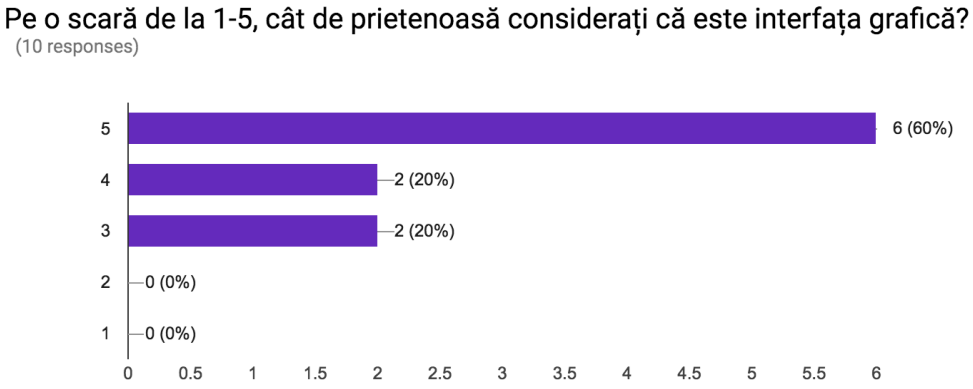


Figura 6.2 Grafic al răspunsurilor de la întrebarea numărul 1 din chestionar

La prima întrebare adresată 60% din participanți au constatat că interfața grafică este suficient de atrăgătoare, în timp ce 40% cred că o îmbunătățire în acest sens ar fi necesară. În partea a doua a chestionarului, unii din utilizatori au descris anumite probleme observate la randarea background-ului în anumite părți ale ecranului. De asemenea, o parte din utilizatori consideră că anumite culori ar putea fi schimbate. Ca o dezvoltare ulterioară se propune îmbunătățirea interfeței grafice.

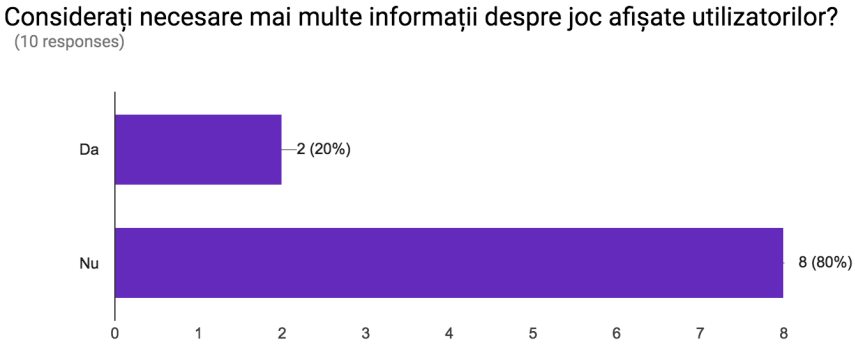


Figura 6.3 Grafic al răspunsurilor de la întrebarea numărul 2 din chestionar

La cea de-a doua întrebare, majoritatea de 80% a participanților consideră că informațiile prezente despre modul de joc sunt suficiente, în timp de 20% sunt de părere că mai multe informații sunt necesare. În acest caz, o dezvoltare ulterioară ar putea fi introducerea informațiilor despre modul în care resursele colectate afectează jocul.

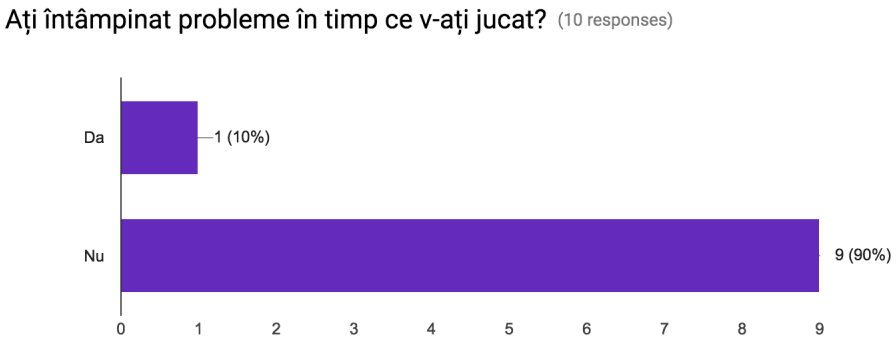


Figura 6.4 Grafic al răspunsurilor de la întrebarea numărul 3 din chestionar

La ultima întrebare adresată se arată că un singur utilizator a avut probleme în timpul jocului. Datorită feedback-ului primit din partea sa, problema a fost rezolvată în versiunea finală a jocului.

Capitolul 7. Manual de Instalare și utilizare

În acest capitol se va prezenta cerințele hardware, manualul de utilizare și instalare al aplicației atât pentru partea de backend cât și pentru partea de frontend.

7.1. Cerințe hardware

În continuarea vor fi prezentate cerințele minime pentru rularea aplicației de backend. Aceste cerințe au fost alese în urma unor serii de teste pe diferite mașini virtuale.

CPU: 1
FREQ: 1,7 GHz
RAM: 512 MB
SSD: 5 GB

7.2. Instalarea și rularea

Pentru instalarea aplicației de backend câteva tooluri sunt absolut necesare și trebuie preinstalate: Git, NodeJS, MongoDB. În continuare se va presupune că aceste tooluri sunt instalate în sistem iar instalarea se poate face în condiții normale.

Pentru simplificarea procesului de instalare codul aplicației a fost hostat pe github. Primul pas este clonarea proiectului de pe github cu ajutorul comenzi „git clone”.

Cel de-al doilea pas este instalarea dependințelor descrise în fișierul package.json. Pentru instalarea lor se va rula comanda „npm install”

Ultimul pas pentru instalare este rularea comenzii „npm start” care va porni backend-ul. O dată pornit, acesta va afișa portul deschis la care se vor putea conecta utilizatorii din browser. În mod implicit este deschis portul 3000, dar se recomandă crearea unui port-forwarding din router care să facă legătura la portul 80 pentru conexiunile din afara rețelei. În final aplicația va fi accesibilă accesând, din browser, următorul url:

http://localhost:3000/

Acest url poate fi deschis în orice browser. Dacă nu au apărut probleme la instalarea aplicației atunci pagina de primire ar trebui să se încarce.

7.3. Manual de utilizare

În continuare va fi prezentat manualul de utilizare al jocului începând cu pagina de primire și autentificare, iar mai apoi prezentarea jocului.

7.3.1. Pagina de primire

Pagina de primire întâmpină utilizatorul cu o interfață prietenoasă, unde utilizatorul este rugat să își introducă numele cu care va fi identificat în joc. După introducerea numelui, utilizatorul trebuie să apese butonul de play pentru a intra în joc alături de alți playeri.

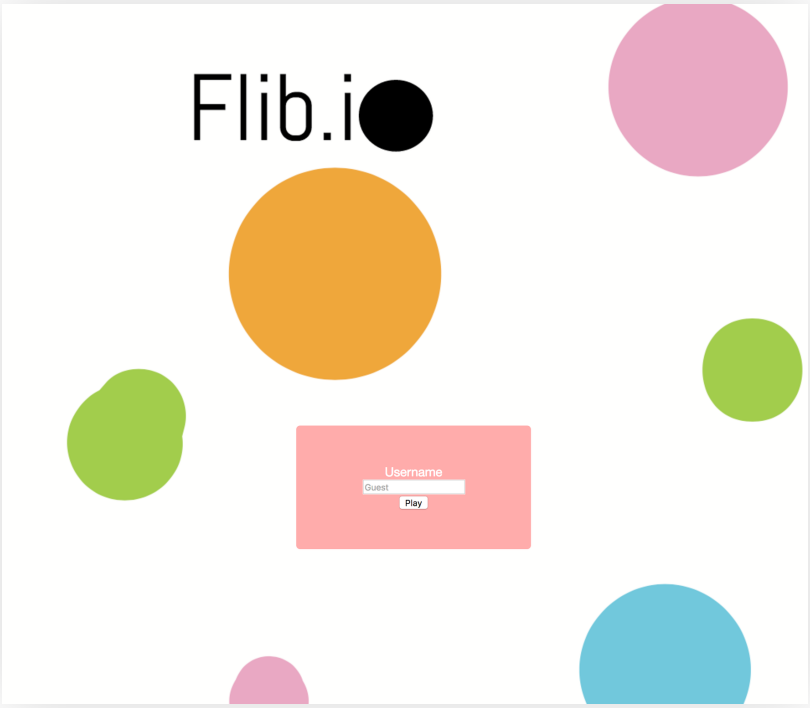


Figura 7.1 Screenshot al paginii de login

7.3.2. Participarea într-o rundă de joc

O dată apăsat butonul de start, utilizatorul va fi redirecționat spre harta de joc. Pentru a efectua mișcări, acesta trebuie să folosească tastele W (sus), S (jos), A (stânga), D (dreapta). De asemenea, pentru lansarea unui proiectil, jucătorul v-a trebui să seteze ținta cu mouse-ul, iar apoi la apăsarea butonului click stânga, proiectilul își va începe deplasarea spre punctul țintit. În partea din dreapta jos se pot vedea informații referitoare la poziționarea utilizatorului pe harta de joc.

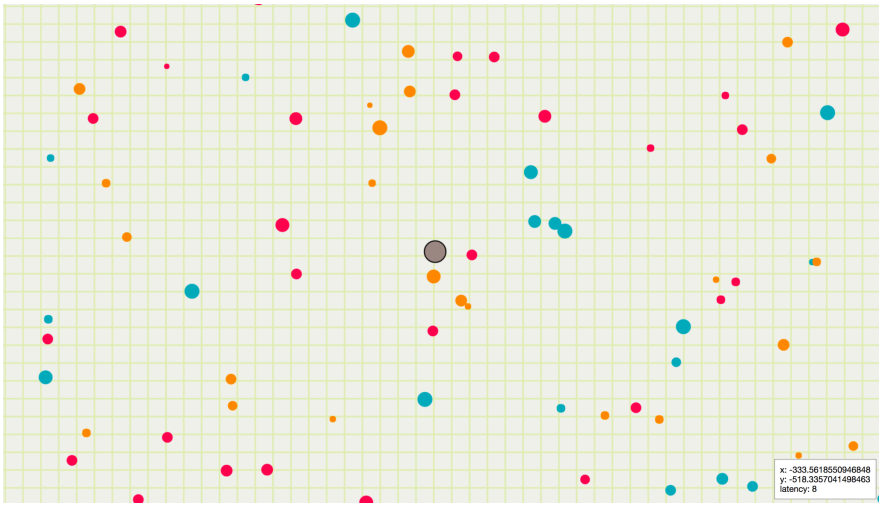


Figura 7.2. Screenshot din interiorul jocului

Se pot observa pe harta de joc o serie de cercuri colorate. Jucătorul este reprezentat printr-un cerc cu bordură de culoare diferită. În principal cercurile roșii, portocalii și cele albastre reprezintă resursele care pot fi colectate.

Capitolul 8. Concluzii

În acest capitol se vor prezenta obiectivele ce au fost atinste în acest proiect, precum și posibilitățile de dezvoltare ulterioare.

8.1. Realizarea obiectivelor propuse

Flib reușește să își atingă țelul, oferind posibilitatea de a permite utilizatorilor să intre într-o competiție cu alți jucători online într-un joc de acțiune unde utilizatorul își va da interesul în a lupta pentru supraviețuire, culegând resursele și distrugând oponenții care l-ar putea împiedica să ajungă cel mai bun.

8.2. Contribuții personale

În vederea dezvoltării acestei aplicații, s-a realizat proiectarea și implementarea unei arhitecturi complexe atât pentru partea de server cât și pentru partea de client.

Pentru partea de backend s-a realizat identificarea celor mai potrivite librării și implementarea algoritmilor care optimizează parcurgerea a miilor de elemente care sunt necesare a fi afișate. O altă contribuție a fost crearea unui sistem de regenerare automată și balansare a resurselor din joc. De asemenea s-au efectuat calcule matematice în vederea verificărilor necesare pentru coliziunile între obiecte și calcularea traictoriei proiectilelor la fiecare tic de ceas. S-a implementat un sistem de verificare a conexiunii, în cazul în care clientul nu răspunde pentru o perioadă de timp acesta este deconectat de la socket. S-a creat un sistem de inteligență artificială care este responsabil cu simularea jocului împotriva unor jucători reali.

Pentru partea de frontend am participat la implementarea designului aplicației, implementarea librăriei CreateJS pentru randarea și mișcarea obiectelor 2D, implemntarea și sincronizarea comunicării cu serverul cu ajutorul librăriei Socket.Io.

8.3. Dezvoltări ulterioare

Datorită faptului că aplicația a fost dezvoltată urmând o serie de design pattern-uri, dezvoltările ulterioare ale aplicației ar fi ușor de implementat. În acest sens se poate menționa că s-a scris codul de la început cu idea de a putea fi implementate funcționalități noi care să atragă cât mai mulți jucători. Codul este foarte ușor de refolosit, de inlocuit. Componentele sunt cât de posibil separate, iar adăugarea componentelor noi necesită foarte puțin efort.

Una din posibilitățile dezvoltării pe viitor este crearea unui sistem care să permită utilizatorilor să își creeze propriile baze de luptă, unde să fie protejați de inamici. În acest sens este nevoie de introducerea unor clase noi pentru construcții și de a crea o legătură între utilizator și construcțiile sale. Afișarea poate fi reprezentată sub forma unui pătrat, ceea ce cu ajutorul librăriei Easel.js este foarte simplă.

O altă posibilitate de dezvoltare ulterioara este posibilitatea de a lăsa utilizatorii să își aleagă camera de joc dorită. Astfel ar fi posibilă creearea de grupuri de prieteni care să joace împreună pe aceeași hartă, fără să le fie asignată aleator.

Adăugarea diferitelor tipuri de arme și proiectile cu puteri speciale, în joc, ar fi o altă dezvoltare ulterioară care ar putea fi implementată cu ușurință.

Pentru partea de inteligență artificială, o dezvoltare ulterioară ar putea fi implementarea unui bot care să poată să și atace utilizatorii de pe hartă, astfel simulând reacțiile unui jucător adevărat. De asemenea se poate implementa un algoritm mai avansat de machine learning care să învețe din diferite situații reale, cum ar proceda un jucător în cazul în care este înconjurat de mai mulți oponenți, iar resursele din jur fiind foarte importante pentru a rămâne în viață.

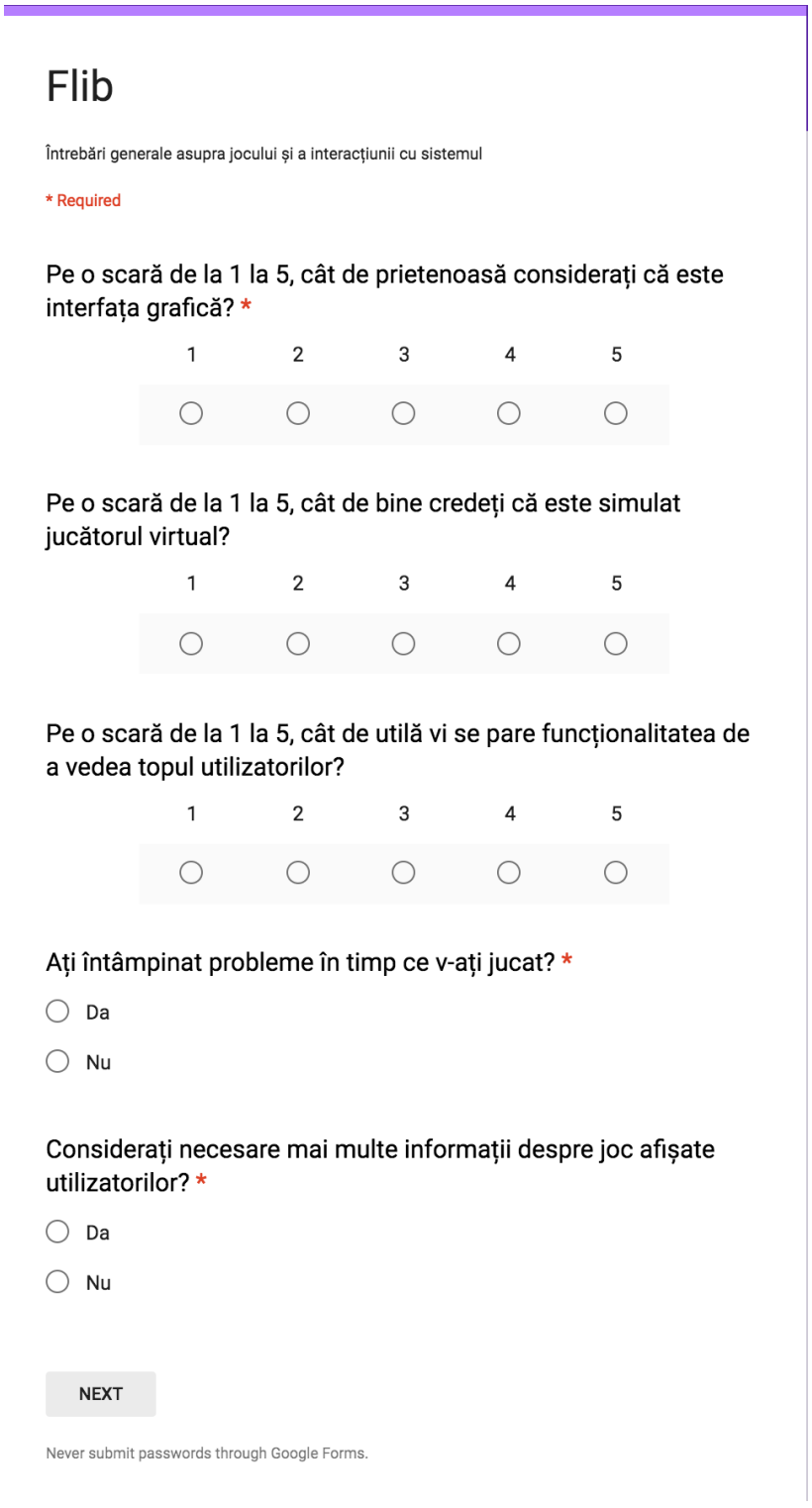
Pentru interfața grafică, o îmbunătățire ar fi dezvoltarea unor aplicații native pentru mai multe sisteme de operare, atât pentru PC cât și pentru alte console, sau telefoane mobile. Acest lucru este posibil deoarece comunicarea cu partea de backend se realizează prin intermediul unor socket-uri.

Bibliografie

[1] History of video games, https://en.wikipedia.org/wiki/History_of_video_games
[2] The Mobile Web Is Dead, It is all about Apps http://www.businessinsider.com/the-mobile-web-is-dead-its-all-about-apps-2014-4
[3] The Use of Mathematics in Computer Games, https://nrich.maths.org/1374
[4] WebSocket, https://en.wikipedia.org/wiki/WebSocket
[5] Socket.IO, http://socket.io/
[6] Agar.io, http://agar.io/
[7] Agar.io Wikipedia, https://en.wikipedia.org/wiki/Agar.io
[8] Slither.io, http://slither.io/
[9] Diep.io, http://diep.io/
[10] Node.js, https://nodejs.org/en/
[11] Node Hero - Understanding Async Programming in Node.js, https://blog.risingstack.com/node-hero-async-programming-in-node-js/
[12] npm (software), https://en.wikipedia.org/wiki/Npm_(software)
[13] Yarn vs npm: Everything you need to Know, https://www.sitepoint.com/yarn-vs-npm/
[14] Express.js, https://en.wikipedia.org/wiki/Express.js
[15] Nodejs Modules, https://nodejs.org/api/modules.html
[16] Jade, https://en.wikipedia.org/wiki/Jade
[17] Simple-Quadtree, https://www.npmjs.com/package/simple-quadtree
[18] Quadtree, https://en.wikipedia.org/wiki/Quadtree
[19] CreateJS, http://www.createjs.com/
[20] EaselJS, http://www.createjs.com/easeljs
[21] TweenJS, http://www.createjs.com/tweenjs
[22] SoundJS, http://www.createjs.com/soundjs
[23] PreloadJS, http://www.createjs.com/preloadjs
[24] Jade, https://www.npmjs.com/package/jade
[25] HTML, https://www.w3schools.com/html/
[26] HTML5, https://www.w3schools.com/html/html5_intro.asp
[27] HTML 5 Canvas, https://www.w3schools.com/html/html5_canvas.asp
[28] CSS, https://www.w3schools.com/css/
[29] Javascript, https://www.w3schools.com/js
[30] DOM, https://en.wikipedia.org/wiki/Document_Object_Model
[31] Ajax, https://www.w3schools.com/js/js_ajax_intro.asp
[32] 6th Edition - EcmaScript2016, https://en.wikipedia.org/wiki/ECMAScript#6th_Edition_-_ECMAScript_2015
[33] JSON, https://en.wikipedia.org/wiki/JSON
[34] MongoDB, https://ro.wikipedia.org/wiki/MongoDB
[35] GIT, https://en.wikipedia.org/wiki/Git
[36] SourceTree, https://en.wikipedia.org/wiki/Atlassian#Products_and_services
[37] Martin Robert Cecil, Clean code: a handbook of agile software craftsmanship.

Upper Saddle River, NJ, 2009
[38] Robert Cecil Martin, https://en.wikipedia.org/wiki/Robert_Cecil_Martin
[39] Webstorm, https://en.wikipedia.org/wiki/JetBrains#WebStorm
[40] Analyze Runtime Performance, https://developers.google.com/web/tools/chrome-devtools/rendering-tools/
[41] Real Time Communications, http://searchunifiedcommunications.techtarget.com/definition/real-time-communications
[42] DigitalOcean, https://www.digitalocean.com/

Anexa 1 - Chestionarul de evaluare al aplicației



Flib

Întrebări generale asupra jocului și a interacțiunii cu sistemul

* Required

Pe o scară de la 1 la 5, cât de prietenoasă considerați că este interfața grafică? *

1	2	3	4	5
○	○	○	○	○

Pe o scară de la 1 la 5, cât de bine credeți că este simulat jucătorul virtual?

1	2	3	4	5
○	○	○	○	○

Pe o scară de la 1 la 5, cât de utilă vi se pare funcționalitatea de a vedea topul utilizatorilor?

1	2	3	4	5
○	○	○	○	○

Ați întâmpinat probleme în timp ce v-ați jucat? *

○ Da

○ Nu

Considerați necesare mai multe informații despre joc afișate utilizatorilor? *

○ Da

○ Nu

NEXT

Never submit passwords through Google Forms.

Flib

Observații

Descrieți ce v-a plăcut la interfața grafică

Your answer

Descrieți ce nu v-a plăcut la interfața grafică

Your answer

Sugestii pentru dezvoltări ulterioare

Your answer

BACK SUBMIT

Never submit passwords through Google Forms.

Anexa 2 - Listă de figuri

Anexa 3 - Listă tabele

Anexa 4 - Glosar de termeni

HTML	Hyper Text Markup Language
CSS	Cascading Style Scripting
JS	Javascript
JSON	JavaScript Object Notation
JSONP	JSON with Padding
IDE	Integrated Development Environment
AJAX	Asynchronous JavaScript and XML
DOM	Document Objet Model
HTTP	HyperText Transfer Protocol
Bot	Program care executa diferite task-uri automat
API	Application Programming Interface