Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Počítačová hra využívající Websocket

2017

Tomáš Dusík

Vedoucí práce: Mgr. Tomáš Kühr, Ph.D.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Tomáš Dusík
Název práce:	Počítačová hra využívající Websocket
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2017
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	Mgr. Tomáš Kühr, Ph.D.
Počet stran:	32
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Tomáš Dusík
Title:	A game using websockets
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2017
Study field:	Applied Computer Science, combined form
Supervisor:	Mgr. Tomáš Kühr, Ph.D.
Page count:	32
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cílem práce je implementovat prohlížečovou hru, která funguje v reálném čase pomocí technologie websockets. Dále uvádím zástupce podobných her a popisuji jakým způsobem se ovládá hra a představuji grafické rozhraní. Programátorská dokumentace se zabývá nejdůležitějšími implementačními detaily aplikace a zvolenými technologiemi. Závěr práce je věnovaný zhodnocení výsledků řešení a možných vylepšení aplikace. Výsledná aplikace implementuje technologii websockets a umožňuje hraní pro více hráčů, kteří spolu mohou v reálném čase bojovat a komunikovat.

Synopsis

The goal of the thesis is to implement a browser game that works in real time using websockets. I am presenting a representative of similar games and describe how the game is controlled and introducing a graphical interface of the game. Programming documentation deals with the most important implementation details of the application and the technologies which has been chosen. The conclusion of the thesis is devoted to evaluation of the results of the solution and possible enhancements of the application. The resulting application implements websockets and allows a lot of players fighting with each other and communicate in real time.

Klíčová slova: websockety; online hra; prohlížečová hra;

Keywords: websockets; online game; browser game

Děkuji své přítelkyni Ing. Martině Goldové a vedoucímu své práce Mgr. Tomáši Kührovi, Ph.D.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Úvod

Tato práce se zaměřuje na implementaci technologie websockets v rámci webové aplikace (MMORPG hry). V této práci popisuji nejdůležitější herní prvky a implementační detaily důležitých algoritmů, které byly použity. Grafické rozhraní je implementováno podle herních standardů, které jsem za roky hraní her nastudoval a jaké bych si jako hráč představoval. Aplikace je lokalizována do angličtiny, vybral jsem tak z důvodu rozšířenosti jazyka po celém světe.

1.1 Cíl práce

Cílem této práce je navrhnout a implementovat webovou aplikaci (hru), která pracuje v reálném čase. Implementovat technologii websockets pro základní funkce jako je chat a interakci mezi hráči. Jedná se především o souboj mezi hráčem nebo počítačem, který je nejzásadnější funkcionalitou celé hry. Vytvořit hráčské atributy a základní předměty, které budou pomáhat hráči v souboji.

1.2 MMORPG

MMORPG je typ počítačových online her obrovského počtu hráču s RPG prvky. MMORPG hry se odehrávají zpravidla ve fiktivním světě, kde hráč hraje za hrdinu [1]. Tento typ her je určen pro velké množství hráčů, kdy je připojení zpravidla zprostředkováno skrze Internet. Hry tohoto typu se vyvinuly z textových RPG online her, tzv. MUDů [6]. Termín MMORPG vymyslel Richard Garriott, tvůrce Ultimy Online [7].

1.3 Websockets

Websockets je počítačový komunikační protokol poskytující plně duplexní komunikační kanál přes jediné TCP připojení. Protokol WebSocket byl standardizován IETF jako RFC 6455 v roce 2011 a jeho WebSocket API je standardizováno W3 [11]. Websockets jsou obdobu **síťových socketů**, které byly představeny pro webové aplikace.

S touto technologii lze napsat webovou aplikaci, kde klientská aplikace v prohlížeči může navázat obousměrné spojení se serverem a po navázání tohoto spojení si mohou vyměňovat informace v reálném čase. Vychází z technologie, jako AJAX. Ajax je asynchronní jednorázový HTTP požadavek na server. Websockets nabízí programátorům to, co AJAX technologie nedokázala: Jednoduché rozhraní pro navázání spojení a následně vzájemnou výměnu zpráv mezi klientem a serverem [10].

1.4 Stávající aplikace

Hry typu MMORPG už existují i pro prohlížeče. Většina těchto typů her, ale nejsou v reálném čase. Po prozkoumání těchto her jsem si vybral **sfgame.com**

[17]. Jedná se o flashovou hru typu MMORPG, která je určena pro prohlížeče. Tato hra není v reálném čase (neimplementuje websockets), to znamená, že souboj, který je základním interaktivním prvkem celé hry, funguje tak, že se spočítají atributy obou hráčů a ten, který má víc, zpravidla vyhrává (při výpočtu ještě hraje důležitou roli náhoda). **sfgame** má velmi rozsáhlou herní komunitu a je známá po celém světě.

1.5 Důvod pro vývoj hry

Prohlížečové hry typu MMORPG existují už i pro prohlížeče. V drtivé většině takových her chybí implementace technologie, která by umožnila hrát v reálném čase, a tedy umožnit hráči lepší zážitek ze hry. Proto jsem se rozhodl implementovat prohlížečovou MMORPG hru, která umožní bojovat a komunikovat s ostatními hráči v reálném čase.

1.6 Popis řešeného problému

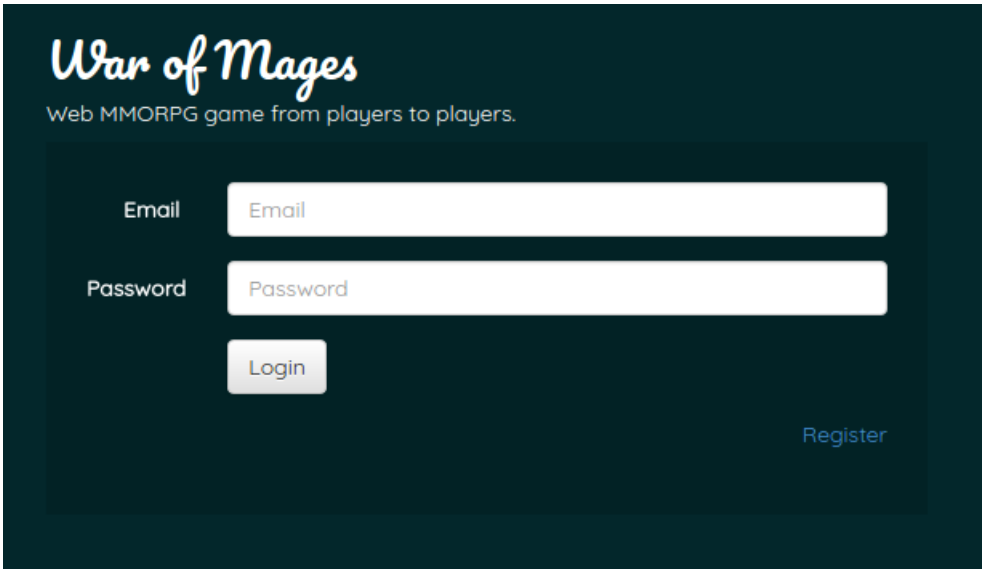
V první fázi je potřeba prostudovat herní doménu a to je v mém případě MMORPG [1] svět a prohlížečové hry. Jak chci aby můj herní svět vypadal, jaké hrdiny budu vytvářet, a jaké herní atributy budu implementovat. V další fázi přichází studie proveditelnosti. Lze pracovat s objektovým paradigmatem. V mém případě to může být javascript (ecma script 2015) jak na serverové tak i klientské části. Pro komunikaci client-server v realném čase lze použít technologii websockets [10] za pomocí knihovny Socket.io [5]. Dále jsem si definoval funkční a mimofunkční požadavky pro aplikaci (aplikace bude lokalizována do angličtiny, server bude umístěn v České republice, ...). Systémové požadavky jsou nenáročné vzhledem k tomu, že se jedná o webovou aplikaci. Server, na kterém poběží "backendová"aplikace, může běžet na platformách jako Linux, Mac i Windows. V předposlední fázi jsem vytvořil návrh databáze, dále návrh serverové a klientské aplikace. Poslední fáze byla určena pro vývoj a testování všech aplikací.

2 Uživatelská dokumentace

V této části představuji funkci grafického rozhraní v mé aplikaci, které je specifické pro prohlížečové hry. Uvádím důležité herní prvky a návod k použití.

2.1 Spustění aplikace

Webová aplikace je umístěna na adrese http://46.28.109.174:8000. Po načtení všech javascriptových skriptů, umístěných na stránce se nám objeví úvodní scéna hry, která obsahuje přihlašovací dialog. Pro přístup do hry potřebujeme mít herní účet, který si musíme vytvořit a nebo můžeme použít testovací účet **test@test.cz** s heslem **test**.



Obrázek 1: Úvodní přihlašovací obrazovka

2.1.1 Registrace

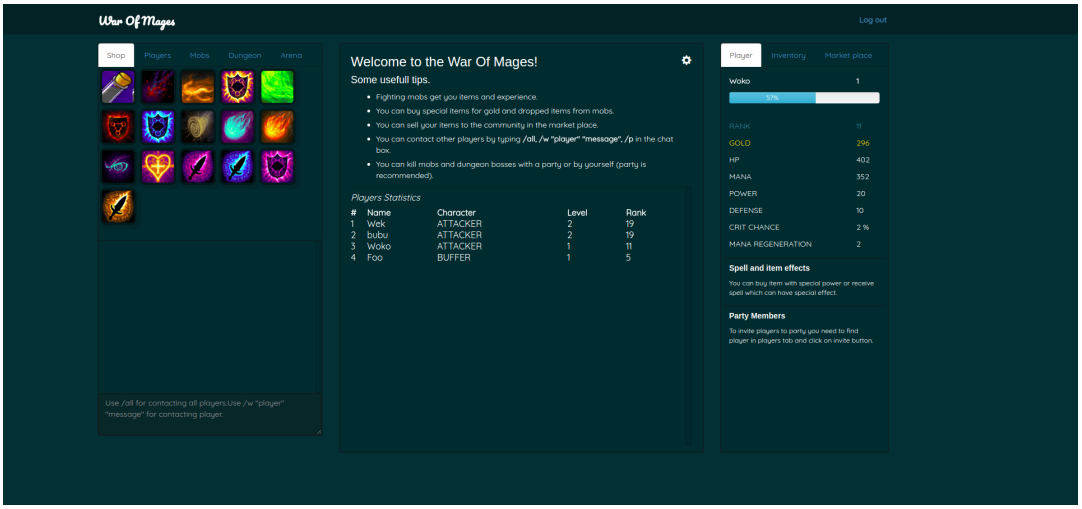
Pro vytvoření herního účtu musíme kliknout na tlačítko **"Register"** a vyplnit základní informace jako je **email** - pod kterým se následně můžeme přihlásit, **nickname** - přezdívku pod kterou budeme vystupovat ve hře, **password** - heslo, které s pomocí emailu slouží pro přístup do hry a nakonec **race** - rasu, za kterou budete hrát ve hře. Na výběr jsou dvě rasy, **Attacker** - útočník, který má převážně útočná kouzla a **Buffer** - pomocník, který má převážně ochranné a podpůrné kouzla, které může vykouzlit na sebe nebo na jiné hráče.

2.1.2 Příhlášení

Do hry se můžeme příhlásit pomocí emailu a hesla, které jsme zadali při registraci. Po úspěšném přihlášení se zobrazí hlavní scéna aplikace.

2.2 Hlavní herní scéna

Hlavní herní scéna po přihlášení obsahuje 3 panely. **Levý panel** obsahuje záložky, které umožňují zabalit další ovládací prvky a ušetřit spoustu místa. Nedílnou součástí levého panelu je také chat sloužící pro komunikaci s herní komunitou. **Prostřední panel** obsahuje základní statistiky o hře a další informace. Zde se v případě souboje vykreslí scéna pro jeho obsluhu. **Pravý panel**, obsahuje záložky.



Obrázek 2: Hlavní herní scéna aplikace

2.3 Ovládání aplikace

Základní ovládací prvky mé hry jsou klávesnice a myš. V této kapitole uvedu, při kterých situacích se používá konkrétní zařízení.

2.3.1 Pomocí myši

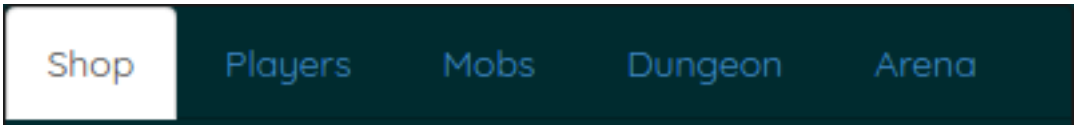
Hra je určená pro prohlížeče, myš je tedy intuitivně nedílným (ne-li nejdůležitějším) ovládacím prvkem mé hry. **Levým kliknutím** vybíráme a používáme interaktivní prvky, které jsou po najetí myší zvýrazněny. Například hráče, kouzla a tlačítka.

2.3.2 Pomocí klávesnice

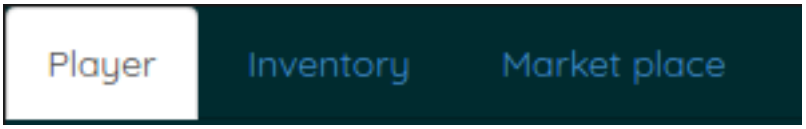
Klávesnice slouží jak pro komunikaci s hráči pomocí chatu, tak i jako alternativa pro použití kouzel stisknutím klávesy, která je zobrazena na kouzlu. Kouzla lze vybírat i levým kliknutím.

2.4 Záložky

Jelikož je aplikace poměrně rozsáhlá, musím vyřešit, jak využít dostupné místo efektivně a podle herních standardů pro prohlížečové hry. Proto jsem vybral záložky. Do každé se dá uschovat poměrně velké množství ovládacích a informačních prvků. Pravá záložka je určena pro zobrazení informací o hráčských atributech a předmětů. Levá záložka zobrazuje veškerou funkčnost, která hra umožňuje. V této kapitole představuji jednotlivé záložky a jejich využití v mé aplikaci.



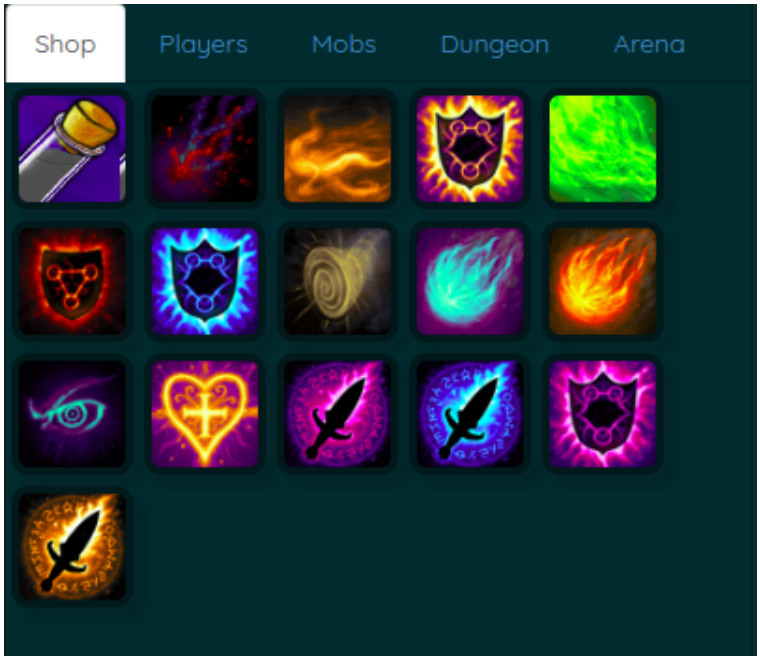
Obrázek 3: Levé záložky



Obrázek 4: Pravé záložky

2.4.1 Shop

V obchodě si můžeme koupit předměty za zlaťáky, které získáme ze zabití příšer. Některé předměty vyžadují kromě zlaťáků i jiné předměty, které si můžeme koupit a nebo získat ze zabití příšer (popřípadě hráčů). Předměty jsou jak podpůrné (přidávají hráči atributy), tak i obyčejné, které jsou potřeba pro dokončení jiného předmětu. Po najetí myši na předmět jsou vždy zobrazeny detailní informace, například které atributy předmět přidává, jaké předměty a kolik zlaťáků potřebujeme ke koupi. Předmět lze koupit levým kliknutím.



Obrázek 5: Obchod s předměty

2.4.2 Players

V této záložce jsou zobrazeni hráči, kteří jsou připojeni ve hře. V případě, že je některý z hráčů připojen jsou zde dvě možnosti interakce. První je pozvat hráče do skupiny, kde můžeme bojovat s hráči ve skupině proti příšeře, nebo bojovat proti jiné skupině hráčů. Další možnost je poslat hráči žádost o souboj. Pokud hráč přijme žádost o souboj, inicializuje se mezi námi souboj a můžeme používat kouzla proti sobě a také používat podpůrná kouzla na sebe.

2.4.3 Mobs

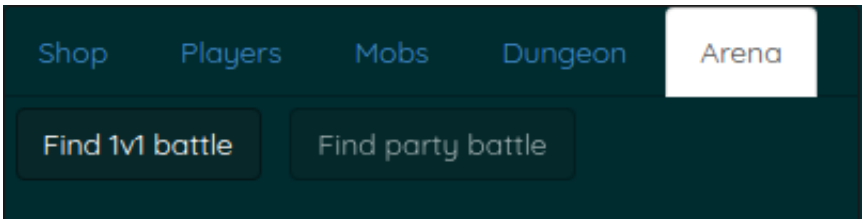
Zde najdeme seznam příšer, na které můžeme zaútočit levým kliknutím myši. Čím je větší úroveň příšery, tím ji bude těžší porazit, ale dostanete za ni lepší odměnu formou zlaťáků nebo předmětů. Po najetí myši na konkrétní příšeru se zobrazí její informace, například úroveň a předměty, které můžeme dostat za zabití příšery.

2.4.4 Dungeon

Tato záložka obsahuje speciální typ příšer, které lze nejlépe porazit pokud jsme ve skupině s dalšími hráči. Zde platí stejné pravidla jako v záložce **Mobs**.

2.4.5 Arena

Aréna slouží pro vyhledání souboje. Jsou zde dvě možnosti, jak můžeme najít svého soupeře. První možnost je najít souboj 1 hráč proti 1 hráči nebo skupina proti skupině. V případě možnosti skupina proti skupině musíme být intuitivně součástí nějaké skupiny, aby byla tato funkcionalita zpřístupněna.



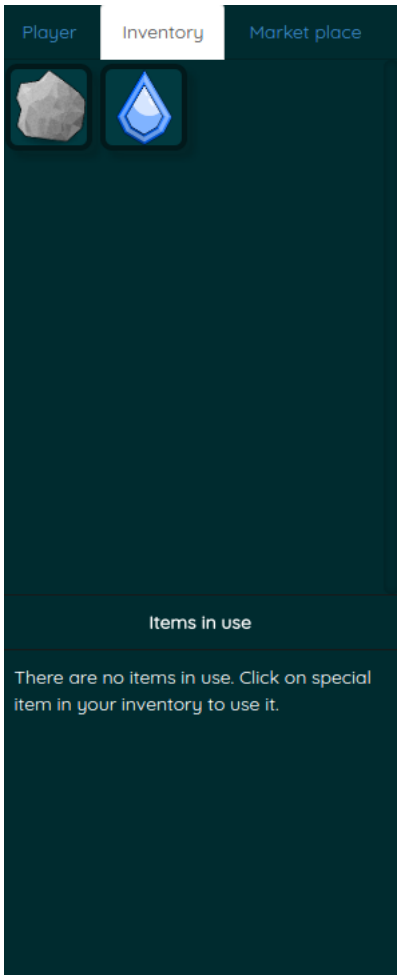
Obrázek 6: Aréna

2.4.6 Player

V této záložce nalezneme veškeré informace o **hráčských atributech**, **speciálních efektech**, které mohou na hráče působit a v neposlední řadě informace o **skupině**, konkrétně kteří hráči jsou momentálně ve skupině s námi.

2.4.7 Inventory

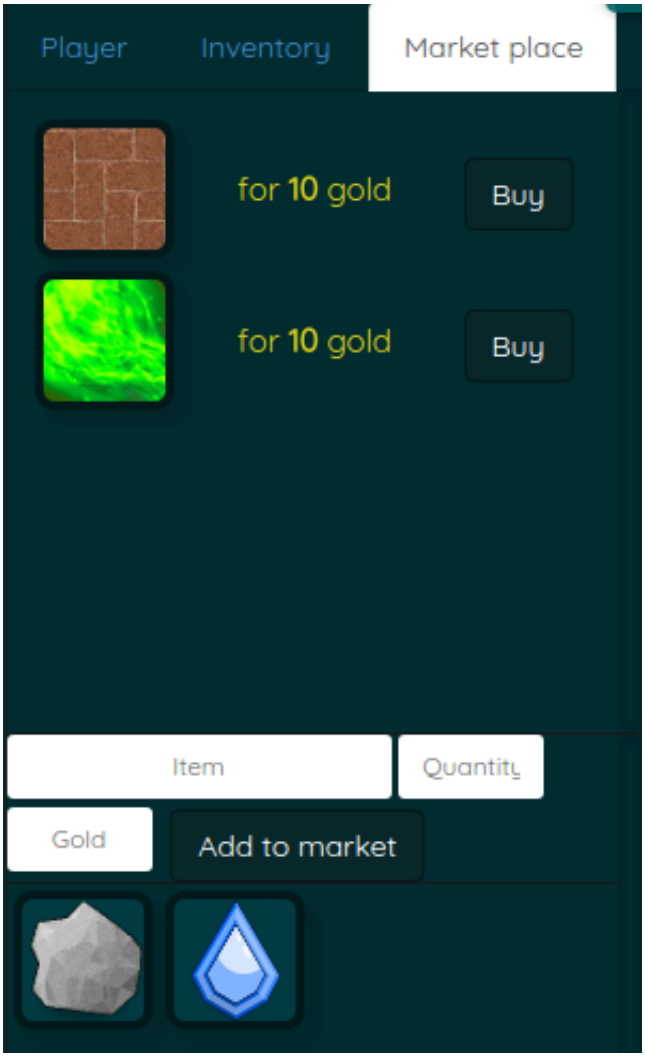
V inventáři máme seznam předmětů, které jsou k dispozici. Lze je získat z obchodu a nebo z úspěšně poražených příšer. Některé předměty lze použít tak, že na ně klikneme levým tlačítkem myši, zmizí nám z inventáře a objeví se v sekci **Items in use** neboli předměty, které jsou momentálně používány. Předměty, které jsou v sekci **Items in use** přidávají atributy, které nám pomáhají v souboji. Můžeme jich zde mít maximálně **8**.



Obrázek 7: Inventář

2.4.8 Market Place

Na trhu můžeme nakupovat předměty, které prodávají jiní hráči. Můžeme sami prodat některý z našich předmětů. Stačí vybrat levým kliknutím předmět, který chceme prodat a zadat jaké množství za jakou cenu.



Obrázek 8: Trh

2.5 Vyhledání souboje

Máme 4 možnosti souboje. **Hráč proti hráči**, **skupina proti skupině**, **hráč proti příšeře**, anebo **skupina proti příšeře**. V této kapitole popíši jakým způsobem se dá uskutečnit souboj ke každé z možnosti.

2.5.1 Proti hráči

Tato možnost lze uskutečnit 2 způsoby. Jedním z nich je vyhledat konkrétního hráče v záložce **Players** a poslat vybranému hráči žádost o duel tlačítkem **Send duel request** a počkat, dokud hráč žádost nepřijme. Dalším ze způsobů je vyhledat náhodný souboj v záložce **Arena**, kde pomocí tlačítka **Find 1V1 battle** můžeme vyhledat protivníka, který hledá souboj stejným způsobem jako my.

2.5.2 Skupina proti skupině

Pro tuto možnost musíme být členem nějaké skupiny. To znamená, že musíme poslat a nebo přijmout pozvání do skupiny a následně v kartě **Arena** kliknout na tlačítko **Find party battle** a počkat než jiná skupina udělá to samé, aby se inicializoval souboj.

2.5.3 Hráč proti příšeře

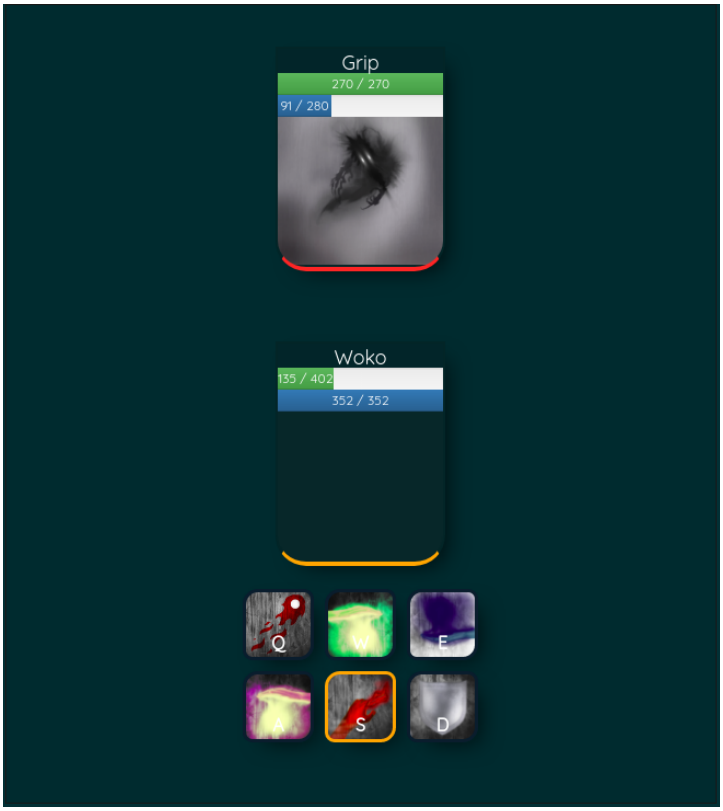
Pokud chceme bojovat proti příšeře, stačí v kartě **Mobs** najít konkrétní příšeru a pomocí levého kliknutí myši inicializovat souboj.

2.5.4 Skupina proti příšeře

Toto je totožné s možností **Hráč proti příšeře**, pouze musíme být ve skupině.

2.6 Souboj

Jak jsem již zmiňoval, souboj je jeden z nejdůležitějších prvků hry. V této kapitole rozeberu důležité ovládací prvky v rámci souboje a upřesním některé důležitější pojmy.



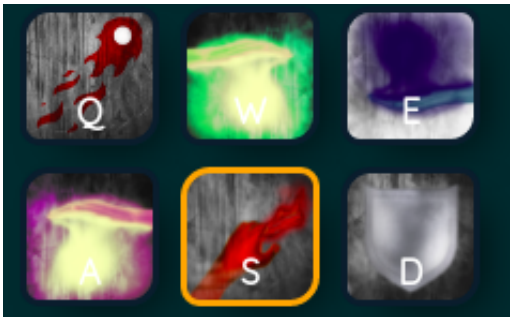
Obrázek 9: Scéna souboje s příšerou

2.6.1 Výběr hráče

Mezi důležité faktory souboje patří také cíl, tedy hráč, na kterého chceme vyslat nějaké kouzlo. Výchozí vybraný cíl je náš nepřítel. Výběr cíle lze uskutečnit pomocí levého kliknutí myši na konkrétního hráče na scéně.

2.6.2 Ovládání kouzel

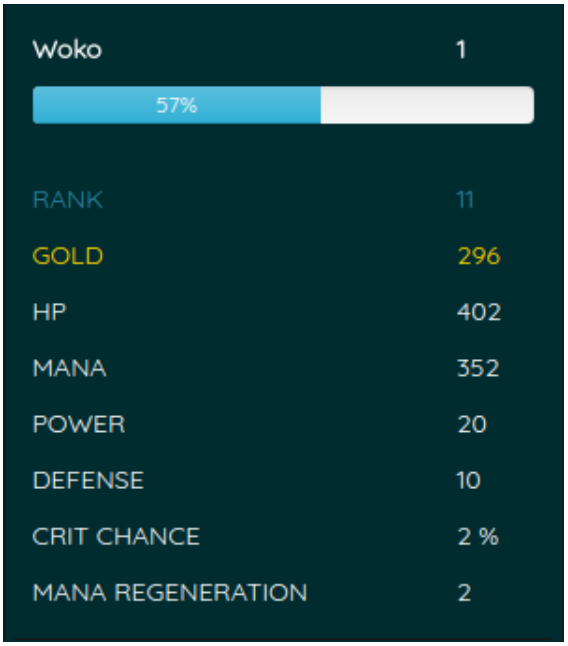
Kouzla můžeme vyslat pomocí kláves **Q, W, E, A, S, D** nebo levým kliknutím myši na vybrané kouzlo. Jelikož je výchozí vybraný hráč náš nepřítel, nemusíme vybírat hráče a můžeme používat útočná kouzla ihned. V případě že chceme použít nějaké kouzlo, které není útočné (přidává nám některé atributy nebo nás léčí), musíme vybrat hráče z naší skupiny anebo sami sebe.



Obrázek 10: Lišta s kouzly

2.6.3 Atributy hráčů

Základním stavebním kamenem souboje jsou právě **atributy**. Atribut **power** udává jakou sílu mají naše kouzla, atribut **defense** nás zase chrání před kouzly od nepřítele. Našim cílem je tedy mít co největší počet atributů, které nám dodávají hlavně předměty, které si můžeme zakoupit v obchodě, anebo je získáme za zabití příšery.



Obrázek 11: Atributy hráče

3 Programátorská dokumentace

V této části popisuji technologie použité na serverové a klientské části. Uvádím, jakým způsobem se zpracovávají hesla od uživatelů a jak probíhá komunikace client-server.

3.1 Serverová část

Server musí obsluhovat požadavky od klienta a umět na ně správně reagovat, aby je klient mohl korektně interpretovat. Proto je potřeba zvolit správné technologie, které nám toto umožní. V této části rozebírám implementační detaily důležitých algoritmů, technologie a jejich využití.

3.1.1 Použité technologie a frameworky

Tato část je věnovaná technologiím a frameworkům, které jsou použity v mé aplikaci. Popisuji základní informace a využití v architektuře aplikace.

3.1.2 Node.js

Je asynchronní prostředí pro javascript, které je postaveno na **V8** enginu běžící i v prohlížeči **Google chrome**. Je navrženo tak, aby se pomocí něho dali škálovat

síťové aplikace [2]. Pomocí Node.js spouštím javascriptovou aplikaci, která běží na serveru a obsluhuje klientské požadavky.

3.1.3 MongoDB

Potřebuji ukládat informace o hráči, například jeho atributy, předměty v inventáři a spoustu dalších informací. Pro tyto účely jsem zvolil databázi **MongoDB**. Jedná se o tzv. nerelační dokumentovou databázi, která ukládá data jako **JSON-like** dokumenty. Je škálovatelná a podporuje indexy [3].

3.1.4 Mongoose

Mongoose slouží pro vytváření databázového modelu (schématu) například pro entitu hráče, které nám zajišťují typovou kontrolu ukládaných dat i větší bezpečnost pro práci s daty [4].

```
1  // příklad vytváření modelu pro entitu hráče pomocí mongoose schéma
2  var mongoose = require('mongoose');
3  var Schema = mongoose.Schema;
4
5  var playerSchema = new Schema({
6   name: String,
7   level: String,
8   attributes: {
9    mana: Number,
10   hp: Number,
11   strength: Number,
12   defense: Number
13  },
14  created: Date,
15  isAdmin: Boolean
16 });
```

Zdrojový kód 1: Mongoose schema

3.1.5 Socket.io

Zprostředkovává komunikaci v reálném čase, která je založená na událostech [5]. Ve hře se vyskytuje nemalé množství případů užití, kdy je potřeba komunikovat jak se serverem, tak i s klientem v reálném čase. Pro tyto potřeby využívám knihovnu **Socket.io**, která tvoří obal nad nativní implementací websockets v javascriptu. Využívám veřejného rozhraní knihovny. Ve většině případech se jedná o metody **emit**, **broadcast** a obsluhovací metodu **on**.

```
1 // příklad metody emit
2 var socket = io('http://foo');
3 socket.emit('some-message', 'Hello server!');
```

Zdrojový kód 2: Poslání zprávy z klienta na server

```
1 // příklad metody on
2 var io = require('socket.io')(app);
3 io.on('connection', function (socket) {
4  socket.on('some-message', function (data) {
5    console.log(data); // Hello server!
6  });
7 });
```

Zdrojový kód 3: Obsluha zpráv na serveru

3.2 Kryptografie a autentifikace

K autentifikaci uživatele je zapotřebí **email** a **heslo**, které uživatel zadal při registraci. Nicméně, aby heslo bylo chráněno a neukládalo se v „plain textu“ používáme kryptografickou hashovací funkci **SHA256** [8]. V tomto případě nemá útočník, při uniknutí databáze, k dispozici hesla uživatelů. Toto řešení je funkční, ale není dostatečně efektivní protože v případě, že dva uživatelé zadají stejné heslo, výsledny hash je stejný. V tomto případě stačí útočníkovi zjistit jaký řetězec se skrývá pod prvním hashem a má vyhráno.

3.2.1 Salt

K tomu aby se z obyčejného zahashované hesla stalo unikátní potřebuji přidat „sůl“ neboli **salt**. Má jediný význam, a to zmenšit pravděpodobnost úniku hesla. [9] Respektive potřebuji docílit toho, aby dvě hesla neměla výsledný hash stejný. Výsledný hash bez saltu se může objevit v některých už spočítaných tabulkách, které využívá útočník k zjištění hesla. V takovém případě je možnost zjištění hesla z uniklé databáze stále vysoce pravděpodobná v případě, že si uživatel zvolil jednoduché heslo.

V praxi se dost často používají ještě tzv. „**salt tabulky**“, kde ke každému heslu je vazba na náhodně vygenerovaný salt. V mé práci salt vytvářím trochu netradičně. Využívám skutečnosti, že email uživatele, který se používá pro autentifikaci, musí být unikátní. Z toho důvodu výsledná funkce, která vytváří hash pro vstupy **email, heslo**, vypadá následovně.

```
function generateHash(email, password) {
  return sha256(email + ':' + password);
}
```

Zdrojový kód 4: Výsledná hashovací funkce pomocí SHA256 se saltem

V případě použití funkce **generateHash** vygeneruji vždy unikátní hash přesto, že dva uživatele zadají stejné heslo, protože každý uživatel vlastní jinou emailovou adresu.

3.2.2 Perzistence hesel

Hesla registrovaných uživatelů musím ukládat do perzistentního úložistě. K tomuto účelu používám **MongoDB**. Databáze MongoDB pracuje s kolekcemi, proto mám vytvořenou kolekci uživatelů, kde každý jednotlivý dokument v kolekci představuje právě jednoho uživatele.

```
{
 "_id": "5932c327f65ca6409437840a",
 "name": "Ivos",
 "email": "ivos@ivos.cz",
 "password": "8bcd76c85f795172e8ed05508a575924a26a..."
}
```

Zdrojový kód 5: Ukázka JSON-like dokumentu v kolekci uživatelů

Heslo se neukládá přímo (v plain textu), ale ukládám výstup funkce **generateHash**. Tento přístup mi zajistí dostatečnou ochranu v případě uniknutí databáze.

3.2.3 Autentifikace

Autentifikace uživatele probíhá při přihlašování na úvodní obrazovce aplikace. Uživatel pošle událost o přihlášení na server, která obsahuje **email a heslo**.

```
var socket = io('http://foo');
socket.emit('LOGIN', {
  "email": "ivos@ivos.cz",
  "password": "test"
});
```

Zdrojový kód 6: Poslání událostí o přihlášení z klienta na server

Server obdrží událost o přihlášení, vygeneruji si hash pomocí funkce **generateHash('ivos@ivos.cz', 'test')** a hledá v kolekci uživatelů uživatele, který má email **ivos@ivos.cz** a heslo, které se rovná výstupu z hashovací funkce. V případě nalezení právě jednoho výskytu pošle uživateli potvrzující zprávu o úspěšném přihlášení nebo selže a pošle zprávu o nezdařeném přihlášení.

3.3 Uživatelská část

K implementaci uživatelského rozhraní jsem využil framework **Angular**, protože se pro tento typ her hodí a má rozsáhlou vývojářskou komunitu. V této části práce popíši, jakým způsobem funguje a představím některé implementační detaily klientské aplikace.

3.3.1 Angular

Jedná se o framework, pomocí kterého se snadno vyvíjí webové aplikace. I přes velké množství funkcí které Angular umožňuje, považuji **data binding** za základní stavební kámen celého frameworku. Jedná se o mechanizmus pro koordinaci části šablony a interního modelu. Přidáním speciální direktivy z angularu do šablony, můžu spojit určitou část (třeba hodnotu prvku formuláře) s interním modelem (například s proměnnou). V případě změny hodnoty proměnné nebo změny hodnoty ve formulářovém prvku, vždy budou hodnoty synchronizovány [16].

Bez data bindingu bych musel řešit synchronizaci šablony (html) a interního modelu aplikace. Například v případě, že se mi do inventáře vloží nový předmět, musel bych projevit změnu jak v interním modelu aplikace, tak ještě překreslit šablonu. To byl ještě triviální případ. Pokud dojde k souboji, kdy je potřeba aktualizovat několik interních modelů pro hráče ve skupinách a ještě k tomu udržovat aktuální šablonu aplikace, by bylo určitě poněkud nevhodné.

Angular podporuje šablony, injektování závislostí a spoustu dalších užitečných technologií vhodných pro vývoj webových aplikací, například routování [13].

3.3.2 Komponenty

Komponenty jsou hiearchické a ovládají html (view). Pomocí komponent definujeme aplikační logiku (co se má stát při stisknutí tlačítka). Komponenty jsou

nejpoužívanější funkcionalitou frameworku a umožňují zanořování (komponenta v komponentě) [16].

V aplikaci se vyskytují desítky komponent a každá je využitá pro konkrétní účel. Například komponenta **LoginComponent**, jak už implikuje název, se používá na úvodní scéně aplikace pro přihlašování. Definice této komponenty vypadá následovně.

```
1  export class LoginComponent {
2   email: string;
3   password: string;
4
5   constructor(private loginService: LoginService) { } // injektování
       LoginService
6
7   // tato funkce se vola při stisknutí tlačítka Login
8   login(email: string, password: string) {
9     this.loginService.login(email, password); // byznys logika uvnitř
          login servisi
10  }
11 }
```

Zdrojový kód 7: Definice angularovské login komponenty v typescriptu

3.3.3 Servisy

Servisy jsou další nejpoužívanější funkcionalitou frameworku. Jedná se o třídu s úzce a velice dobře definovaným účelem. Komponenta může konzumovat až několik servis. Doporučená praxe je udržovat byznys logiku právě v této třídě [13].

Naváži na příklad s přihlašováním a vytvořím jednoduchou servisu (třídu), kterou injektuje LoginComponent a volá nad ní metodu **login**.

```
1  export class LoginComponent {
2    constructor(private socket: SocketService) { } // injektování
        SocketService pro komunikaci se serverem
3    // tuto funkcí volá LoginComponent
4   login(email: string, password: string) {
5     this.socket.emit("LOGIN", {email, password}); // pošleme událost
         o přihlášení na server
6   }
7  }
```

Zdrojový kód 8: Definice angularovské login servisy v typescriptu

4 Komunikace mezi klientem a serverem v rámci souboje

Nedílnou součástí hry je samotný souboj jak mezi hráčem, tak i počítačem. V této kapitole rozeberu architekturu a implementaci.

4.1 Vyhledání souboje hráč proti hráči

Souboj lze navázat dvěma způsoby:

1. vyhledat souboj v aréně,
2. zaslat žádost o souboj konkrétnímu hráči.

4.1.1 Vyhledání souboje v aréně

V případě vyhledání souboje v aréně je potřeba vytvořit jednoduchou strukturu (pole) v paměti, do které se budou přidávat hráči, kteří nenašli v poli protivníka podle konkrétních kritérii. V případě nalezení protivníka v poli se mezi hledaným a nalezeným hráčem inicializuje souboj.

```
1  var PlayerStack = {
2    players: [], // pole do, kterého budeme přidávat hráče, kteří nena
         šli shodu
3    addPlayer: function(player){
4      this.players.push(player);
5    },
6    findMatchForPlayer: function(player){
7      for (var i = 0; i < this.players.length; i++){
8        if(isMatchingCriteria(this.players[i], player)){
9          return this.players.splice(i, 1)
10       }
11     }
12   }
13 };
```

Zdrojový kód 9: Pseudokód implementující práci s polem

Implementace inicializace souboje pak vypadá následovně.

```
// obdržena událost vyhledání protivníka
io.on('find-enemy', function(connection){
  var player = getPlayerFromConnection(connection);
  var enemyPlayer = PlayerStack.findMatchForPlayer(player);

  if(enemyPlayer){
    BattleInstance.initializeBattle(player, enemyPlayer);
  } else {
    PlayerStack.addPlayer(player); // přidáme hráče do pole, protože
        nebyla shoda
  }
});
```

Zdrojový kód 10: Inicializace souboje pomocí areny

4.1.2 Zaslání žádosti o souboj konkrétnímu hráči

V případě zaslání žádosti o souboj je třeba odeslat událost o souboji cílovému hráči, ten následně může žádost potvrdit, anebo se žádost automaticky po stanoveném čase zamítne.

```
socket.emit('duel-request', targetedPlayerId);
```

Zdrojový kód 11: Odeslání žádosti o souboj z klienta od hráče X hráči N

Server obdrží žádost o souboji od hráče X a pošle žádost dál hráči N.

```
io.on('duel-request', function(connection, targetedPlayerId){
  var requestedPlayer = getPlayerFromConnection(connection); // hráč
      X
  var targetedPlayer = getOnlinePlayerById(targetedPlayerId); // hrá
     č N

  targetedPlayer.emit('duel-request', requestedPlayer.getId()); //
     zašleme hráči N žádost o souboj od hráče X
});
```

Zdrojový kód 12: Předání žádosti o souboj hráči N na serveru

Hráč N obdrží žádost o souboj od hráče X a přijme ji.

```
1 io.on('duel-request', function(fromPlayerId){
2   socket.emit('accept-duel-request', fromPlayerId); // hráč N přijím
        á souboj od hráče X
3 });
```

Zdrojový kód 13: Přijmutí žádosti o souboj na klientu

Poslední krok je vytvořit na serveru instanci souboje mezi hráčem X a hráčem N.

```
1 io.on('accept-duel-request', function(connection, requestedPlayerId)
      {
2   var targetedPlayer = getPlayerFromConnection(connection); // hráč
        N
3   var requestedPlayer = getOnlinePlayerById(requestedPlayerId); //
        hráč X
4
5   if(requestedPlayer.askedForDuel(targetedPlayer)){
6     BattleInstance.initializeBattle(requestedPlayer, targetedPlayer);
7   }
8 });
```

Zdrojový kód 14: Inicializace souboje

5 Zhodnocení výsledků řešení

V této části vyjádřím své myšlenky a připomínky k výsledné aplikaci a celému vývojovému procesu.

5.1 Splněné požadavky

Implementoval jsem funkční komunikaci v realném čase pomocí technologie **websockets** a knihovny **Socket.IO**. Úspešně se mi podařilo zrealizovat oboustrannou komunikaci pro souboj mezi hráčem a příšerou, hráčem a hráčem a dokonce zaexperimentovat a úspěšně implementovat souboj mezi skupinou a skupinou. Souboj mezi skupinou a skupinou považuji za nejzábavnější prvek ve hře vzhledem k tomu, že jsme s přáteli a kolegy dokázali strávit poměrně velké množství času právě nad toutou funkcionalitou. Cíle, které jsem si na začátku projektu stanovil, tedy považuji za splněné.

5.2 Možná vylepšení a postřehy při implementaci

Do budoucna bych si dokázal představit výběr z většího množství hrdinů, větší množství příšer a vylepšit umělou inteligenci. Umělá inteligence u příšer je mo-

mentálně implementována triviálně, jak to u her typu MMORPG většinou bývá. Stávající implementace spočívá v tom, že po nějakém časovém intervalu příšera zavolá nad protivníkem útočné kouzlo. Do budoucna bych chtěl implementovat lepší inteligenci u příšer, které by mohli umět reagovat na některé situace v souboji. Další návrh na změnu je určitě grafické rozhraní trhu ve hře.

5.3 Nasazení serverové aplikace

Nasazování aplikací pro různá prostředí je v dnešní době velké téma. Snažil jsem se mít takovou strukturu a architekturu aplikace, aby byla snadno k nasazení do ostrého provozu. Důležité je mít správně navrhnuté konfigurační soubory a ukládat do nich HTTP porty, url adresy pro soketovou komunikaci a veškeré hodnoty, které se mohou měnit v závislosti na prostředí. Docílil jsem toho, že mi, s pomocí konfiguračních souborů a verzovacího systému GIT [15], nasazení aplikace na jiný server trvalo pár minut.

Závěr

Prošel jsem jednotlivými fázemi vývoje od prostudování herní domény, až po návrh a implementaci aplikace. Úspešně jsem vytvořil komunikaci mezi klientem a serverem v reálném čase pomocí technologie websockets. Podařilo se mi vytvořit a vyladit různé varianty souboje ve hře, komunikaci mezi hráčemi a autentifikace uživatele do systému. Klientská aplikace umožňuje přihlášení uživatele do hry, fungující trh a souboje mezi různorodými entitami v reálném čase. Cíle, které jsem si definoval na začátku práce jsou splněny.

Conclusions

I have gone through the development stages from studying the game domain to designing and implementing the application. I have successfully created communication between the client and the server in real time using websockets. I have managed to create and fine-tune the various variants of the match in the game, communication between players and user authentication into the system. The client application allows users to log in to the game, a working market, and real-time combat between heterogeneous entities. The goals that I have defined at the beginning of my work are fulfilled.

A Obsah přiloženého CD/DVD

Stručný obsah adresářové struktury a popis důležitých souborů.

`bin/`
: Kompletní adresářová struktura webové aplikace (v ZIP archivu) pro zkopírování na webový server. Adresář obsahuje i všechny runtime knihovny a další soubory potřebné pro bezproblémový provoz webové aplikace na webovém serveru.

`doc/`
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty webové aplikace se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření adresářové struktury pro zkopírování na webový server.

`readme.txt`
: Instrukce pro nasazení webové aplikace na webový server, včetně všech požadavků pro její bezproblémový provoz, a webová adresa, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.

Literatura

[1] MMORPG. Massively multiplayer online role-playing game [online]. [cit. 2017-02-06]. Dostupné z: https://goo.gl/bectLJ

[2] NODEJS. Node.js [online]. [cit. 2017-02-06]. Dostupné z: https://nodejs.org/en/

[3] MONGODB. What is MongoDB [online]. [cit. 2017-02-06]. Dostupné z: https://www.mongodb.com/what-is-mongodb

[4] MONGOOSE. Mongoose ODM [online]. [cit. 2017-02-06]. Dostupné z: http://mongoosejs.com/

[5] SOCKET.IO. Socket.IO [online]. [cit. 2017-02-06]. Dostupné z: https://socket.io/

[6] MUD. Multi User Dungeon [online]. [cit. 2017-02-06]. Dostupné z: https://cs.wikipedia.org/wiki/MUD

[7] ULTIMA. Ultima Online [online]. [cit. 2017-02-06]. Dostupné z: https://uo.com/what-is-uo/

[8] SHA256. Kryptografická hashovací funkce [online]. [cit. 2017-02-06]. Dostupné z: https://sha256.com/

[9] SALT. How does password salt help against a rainbow table attack? [online]. [cit. 2017-02-06]. Dostupné z: https://stackoverflow.com/questions/420843/how-does-password-salt-help-against-a-rainbow-table-attack

[10] WEBSOCKETS. Web Sockets [online]. [cit. 2017-02-06]. Dostupné z: https://www.zdrojak.cz/clanky/web-sockets/

[11] WEBSOCKET WIKI. Web Socket Protokol [online]. [cit. 2017-02-06]. Dostupné z: https://en.wikipedia.org/wiki/WebSocket

[12] SALT. Can you help me understand what a cryptographic "salt" is? [online]. [cit. 2017-02-06]. Dostupné z: https://crypto.stackexchange.com/questions/1776/can-you-help-me-understand-what-a-cryptographic-salt-is

[13] ANGULAR. What is Angular? [online]. [cit. 2017-02-06]. Dostupné z: https://angular.io/docs

[14] ANGULAR DOC. Data binding [online]. [cit. 2017-02-06]. Dostupné z: https://angular.io/guide/architecture

[15] GIT. Verzovací email [online]. [cit. 2017-02-06]. Dostupné z: https://git-scm.com/

[16] SHA256. SHA-256 Cryptographic Hash Algorithm [online]. [cit. 2017-02-06]. Dostupné z: http://www.movable-type.co.uk/scripts/sha256.html

[17] SFGAME. Shakes and Fidget [online]. [cit. 2017-02-06]. Dostupné z: http://www.sfgame.cz/

[18] OPEN GAME ART. Rozsáhlá databáze obrázků a zvuků pro hry zdarma [online]. [cit. 2017-02-06]. Dostupné z: https://opengameart.org/