VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SÍŤOVÁ KOMUNIKACE A KOLABORATIVNÍ SDÍLENÍ DAT PRO STRATEGICKOU HRU NA WEBOVÉ PLATFORMĚ

NETWORK COMMUNICATION AND COLLABORATIVE DATA SHARING FOR STRATEGY GAME ON WEB PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VOJTĚCH KULÍŠEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN PEČIVA, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Kulíšek Vojtěch**
Program: Informační technologie
Název: **Síťová komunikace a kolaborativní sdílení dat pro strategickou hru na webové platformě**
Network Communication and Collaborative Data Sharing for Strategy Game on Web Platform
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte si vybrané metody kolaborativního sdílení dat ve virtuálních scénách. Prostudujte možnosti vytváření síťového spojení na webových technologiích a WebRTC.
2. Navrhněte jednoduchou síťovou strategickou hru na webové platformě. Navrhněte konzistenční model pro kolaborativní sdílení dat v této aplikaci. Speciálně zvažte použití modelů konzistence využívající aktivní replikaci dat.
3. Implementujte navrženou webovou aplikaci. V aplikaci implementujte navržený konzistenční model dat.
4. Ověřte správnou funkčnost aplikace a konzistenčního modelu. Proveďte měření latence a množství přenášených dat. Proveďte měření i za zhoršených síťových podmínek a vyhodnoťte použitelnost aplikace v těchto zhoršených podmínkách.
5. Diskutujte možná budoucí vylepšení. Práci publikujte na internetu pod některou z open-source licencí, nebo zvolte jinou formu prezentace projektu (plakátek, apod.).

Literatura:

- PEČIVA Jan. Active Transactions in Collaborative Virtual Environments. Brno: Faculty of Information Technology BUT, 2007. ISBN 978-80-214-3549-0, dostupné na https://www.fit.vut.cz/research/publication/8571/
- peer.js dokumentace
- dále dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Prototyp běžící aplikace.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Pečiva Jan, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Cílem této práce je seznámit čtenáře se základními metodami kolaborativního sdílení dat, aplikačními rozhraními prohlížečů pro komunikaci přes paketovou síť a vytvořit na základě těchto informací webovou hru, která se bude snažit co nejvíce potlačit latenci sítě.

Abstract

The aim of this work is to acquaint readers with the basic methods of collaborative data sharing, browser application interfaces for packet network communication and create based on this information a web game that will try to suppress network latency as much as possible.

Klíčová slova

verze HTTP protokolu, XHR, Fetch, WebSocket, EventSource, WebRTC, konfigurace kolaborativní scény, systémové hodiny, monotónní hodiny, synchronizace času, iluze nulové latence, problémy konzistence

Keywords

HTTP versions, XHR, Fetch, WebSocket, EventSource, WebRTC, collaborative virtual eviroments configurations, time-of-day clocks, monotonic clock, clock synchronization, latency hiding, consistency issues

Citace

KULÍŠEK, Vojtěch. *Síťová komunikace a kolaborativní sdílení dat pro strategickou hru na webové platformě*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Pečiva, Ph.D.

Síťová komunikace a kolaborativní sdílení dat pro strategickou hru na webové platformě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Jana Pečivy. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Kulíšek
9. května 2022

Poděkování

Děkuji vedoucímu práce doktoru Janu Pečivovi, za vedení této práce a pomáhání při její řešení.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V dnešní době rychlého vývoje webových prohlížečů přichází možnost vytvářet webové hry, které mohou mezi sebou komunikovat na přímo, bez nutnosti přeposílat všechny informace přes server. Díky tomu jsou schopné snížit latenci hry a náklady na provoz. Každopádně převážná většina dnešních populárních webových her tuto možnost nevyužívá, i když by z této metody mohly profitovat.

Lze do budoucna předpokládat, že využití těchto technologií bude čím dál tím častější, jelikož dnes existují hry, které nejsou určené pro webové prohlížeče a nebyly aplikačními rozhraními prohlížečů omezovány a s možností vytvářet přímá propojení profitují a využívají ji již dlouho dobu. Ve webových prohlížečích je tato možnost poměrně krátkou dobu.

V této práci bude popsán vývoj takové webové hry, která bude s možností vytvářet přímá spojení mezi hráči profitovat a bude jejím cílem co možná nejvíce snížit latenci sítě během hraní.

Kapitola 2 seznamuje čtenáře s verzemi protokolu HTTP, dostupnými aplikačními rozhraními se kterými lze komunikovat z webového prohlížeče. Dále popisuje základními metody kolaborativního sdílení dat. V kapitole 3 bude čtenář seznámen s podrobným návrhem hry. Kapitola 4 popíše implementaci aplikace. Výsledky z měření naprogramované hry budou popsány v kapitole 6 a kapitola 7 zhodnotí celkové výsledky této práce.

Kapitola 2

Metody kolaborativního sdílení dat a komunikace webových prohlížečů přes paketovou síť

Následující kapitola vysvětluje a porovnává síťovou komunikaci přes různá aplikační rozhraní implementovaná v prohlížečích webových stránek. Dále porovnává různé konfigurace kolaborativních scén. Vysvětluje, jak se provádí iluze nulové latence. Popisuje, jak funguje synchronizace hodin na protokolu NTP a provádí osvětu některých problémů, které mohou způsobit nekonzistenci kolaborativní scény.

2.1 Rozdíl mezi HTTP a HTTPS protokolem

HTTPS protokol je rozšířený protokol HTTP o šifrování. Pokud bude v práci uveden HTTP protokol, myslí se tím i jeho šifrovaná verze HTTPS.

2.2 Verze HTTP protokolu

HTTP/0.9

Komunikuje na transportním protokolu TCP. Klient odesílá po propojení dotaz, který je jen na jednom řádku. Po přijetí odpovědi ze serveru na dotaz se uzavře TCP spojení. Dotaz je přenášen v řetězci a je pro člověka snadno čitelný. [4]

HTTP/1.0

Umožňuje oproti předchozí verzi posílat v dotazu metadata navíc. [4]

HTTP/1.1

Hlavním vylepšením oproti předchozímu protokolu je udržování TCP spojení po dokončení přenosu dat ve výchozím nastavení. Pokud prohlížeč následně potřebuje stáhnout nová data ze serveru, nemusí vytvářet nové spojení a ušetří tím čas. [4]

HTTP/2

Umožňuje oproti svým předchůdcům provádět na jednom TCP spojení multiplexování. Server může díky tomu na jednom spojení odpovídat na více dotazů klientovi a nemusí čekat na zpracování předchozího dotazu, který provedl. [4]

Tato verze protokolu navíc umožňuje serveru odesílat soubory, na které se klient ještě nedotázal a bude je potřebovat k zobrazení webové stránky. Díky tomu se sníží latence. [4]

Hlavičky jsou v této verzi oproti předchozím verzím posílány v binární podobě, v předchozích verzích byly posílány v řetězci, který byl pro člověka jednoduše čitelný. [4]

HTTP/3

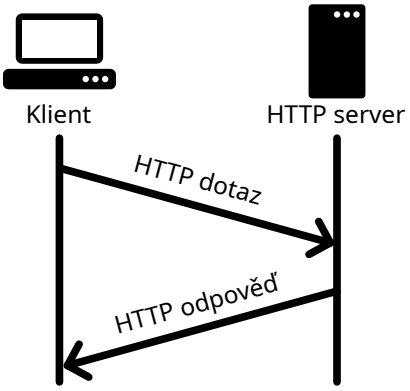
Oproti předchozím verzím místo TCP transportního protokolu využívá protokol QUIC, který je implementován na UDP protokolu. Díky použití protokolu QUIC jsou nižší režie na vytvoření spojení. [1] Protokol HTTP/3 ještě není standardizovaný v době psaní této práce (květen 2022), jak je uvedeno v [1].

2.3 Aplikační rozhraní prohlížečů pro komunikaci přes paketovou síť

XHR, Fetch

XHR je nejstarší aplikační rozhraní implementované v JavaScriptu, pomocí kterého jde komunikovat přes počítačovou síť [7, 8, 9, 10, 11]. Jako první bylo implementováno v prohlížeči Internet Explorer 5 [4]. Fetch je jeho novější nástupce [8].

Aplikační rozhraní XHR se velmi využívá v Asynchronous JavaScript and XML[1] (AJAX) modelu. AJAX model umožňuje vytvářet stránky, které mohou získávat nová data ze serveru bez nutnosti znovu načítat celou stránku. [12]



Obrázek 2.1: Komunikace přes XHR a Fetch

Tyto aplikační rozhraní komunikují se serverem na protokolu HTTP [4]. Stejným způsobem jako prohlížeč získává webové stránky, popřípadě odesílá formuláře. Přes aplikační rozhraní se vytvoří dotaz, ten je následně odeslán na server, kde je vyhodnocen. Ze serveru je doručena odpověď. Obrázek 2.1 ukazuje typickou komunikaci.

[1] https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

Verzi použitého protokolu HTTP a použití nebo nepoužití šifrování, řeší prohlížeč za programátora, není nutné speciálně přizpůsobovat kód pro jednotlivé verze [4].

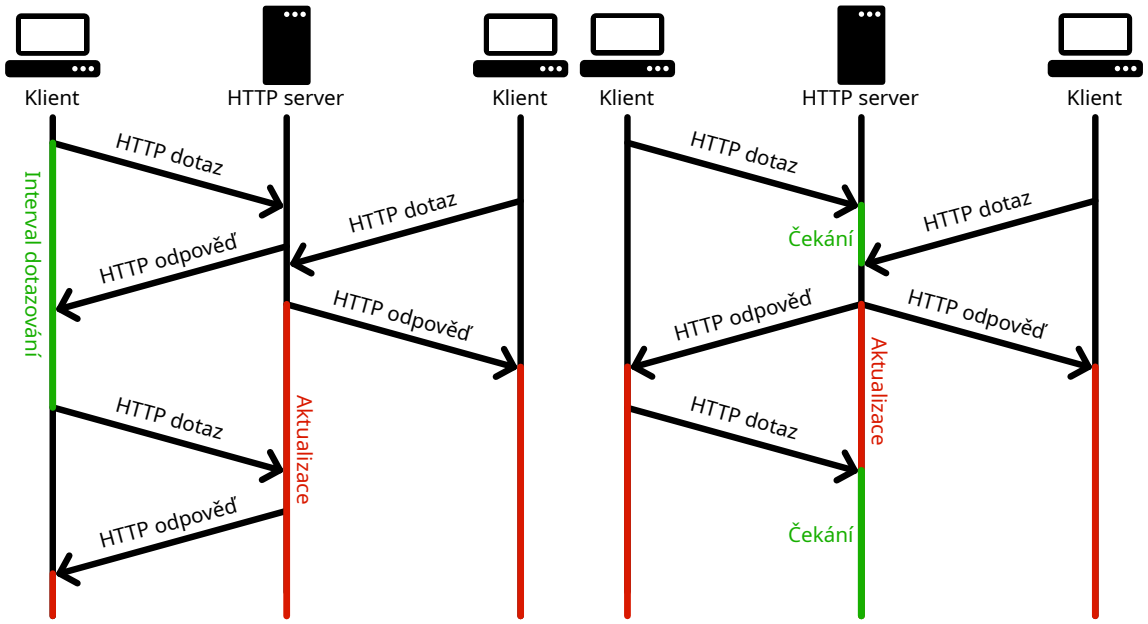
Komunikace vždy probíhá dotazem z prohlížeče a následnou odpovědí na dotaz ze serveru. Server nemůže v tomto případě přímo vytvářet dotazy na klienta, nebo sdělit klientovi aktualizaci některých dat (například aktualizace chatu) bez dřívějšího dotazu z klienta.

Specifikace XHR2 umožňuje průběžně číst přenos po dotazu ze serveru, ale její implementace je velmi neefektivní [4]. Je lepší se této možnosti vyhnout a místo ní použít aplikační rozhraní EventSource, které je zmíněno v této podkapitole níže. Stremování opačným směrem není možné [4].

Nejjednodušší strategie, která se používá k synchronizaci se serverem, je v intervalu se dotazovat na aktualizace serveru (polling) [4]. Server následně odpoví, zda došlo k aktualizaci a navrátí aktualizovaná data, nebo odpoví, že k aktualizaci nedošlo [4]. Tento typ komunikace je vyobrazen na obrázku 2.2a.

Tuto strategii lze vylepšit tím, že server hned neodpoví a vyčkává na aktualizaci dat a následně odpoví, klient hned odešle dotaz poté, co dostane odpověď ze serveru. Tato vylepšená strategie se nazývá Long polling. [4] Pro lepší představu je možná ukázka použití vyobrazena na obrázku 2.2b. Long polling dříve používal Facebook v chatu [4]. Každopádně dnes (5. 4. 2022) na základě vlastního zkoumání jsem zjistil, že používá aktuálně WebSocket pro přenos zpráv. WebSocket je zmíněný níže v této podkapitole. WebSocket je novější než XHR [7, 10].

Každopádně je nutné u pollingu a long pollingu počítat s vyššími režiemi na přenos dat po síti. Jelikož komunikace probíhá na protokolu HTTP je nutné při komunikaci v každém dotazu a odpovědi přenášet hlavičky tohoto protokolu. Seznam položek, které mohou být v HTTP halvičce je k dispozici na stránkách organizace IANA (https://www.iana.org/assignments/http-fields/http-fields.xhtml). V protokolu HTTP 1.1 jsou přenášeny textově a v novějším protokolu HTTP 2 jsou kódovány binárně z důvodu snížení režie [4].



(a) Polling (b) Long polling

Obrázek 2.2: Typy pollingu

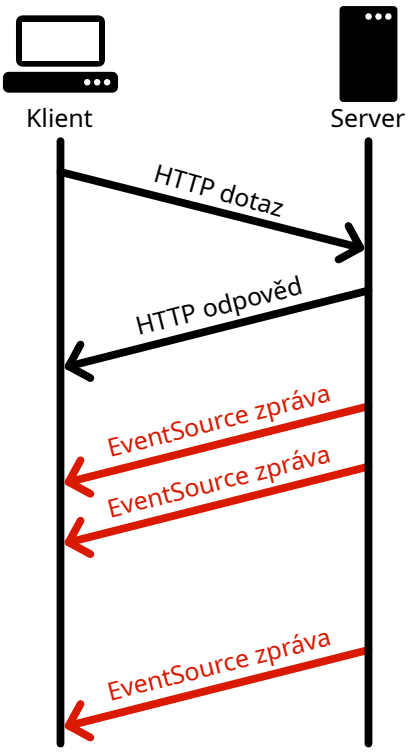
EventSource

Při použití aplikačního rozhraní EventSource prohlížeč na začátku komunikuje přes HTTP protokol a následně přejde na EventSource protokol. Klient může na server posílat data jen na začátku spojení v HTTP dotazu. Po odeslání HTTP hlavičky ze serveru, se už jen odesílají data na klienta, prohlížeč tento přenos průběžně zpracovává. [4] Průběh komunikace je graficky znázorněn na obrázku 2.3.

Poskytuje automatické znovu připojení při pádu spojení. Při znovu připojení může odesílat ID poslední úspěšně doručené zprávy. [4]

Aplikační rozhraní je vhodné používat s protokolem HTTP/2 a novějším. Ve starších protokolech může dojít při otevření mnoha karet k vyčerpání maximálního povoleného množství otevřených spojení (6 spojení) [9]. HTTP/2 a novější používají v rámci jednoho spojení multiplexování, díky kterému jde přes jedno otevřené spojení paralelně odpovídat na více dotazů [4].

Pro použití oboustranné komunikace je vhodnější použít aplikační rozhraní WebSocket. Websocket je uvedené níže v této podkapitole.



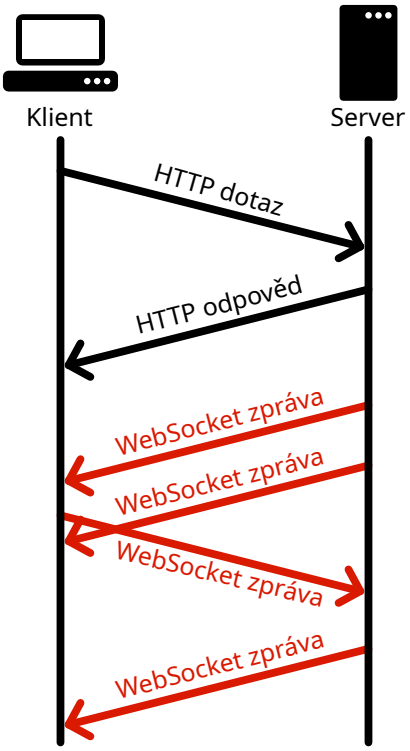
Obrázek 2.3: EventSource

WebSocket

WebSocket je nejbližší aplikační rozhraní k soketům [4]. Umožňuje vytvářet obousměrnou komunikaci se serverem bez nutnosti provádět pooling [10]. Komunikace je vždy bezztrátová a uspořádaná [4].

Na začátku spojení je odeslán dotaz na server v HTTP protokolu o změně protokolu na websocket, následně je ze serveru odeslána odpověď v protokolu HTTP o potvrzení změny,

posléze probíhá komunikace už jen přes websocket protokol. Každý účastník je schopen odeslat data v jakýkoliv čas. [4, 13]



Obrázek 2.4: WebSocket

Dnes je velmi využívaný u počítačových her v prohlížečích. Je například použit ve hrách agar.io[2], deap.io[3], slither.io[4] a krunker.io[5]

WebRTC

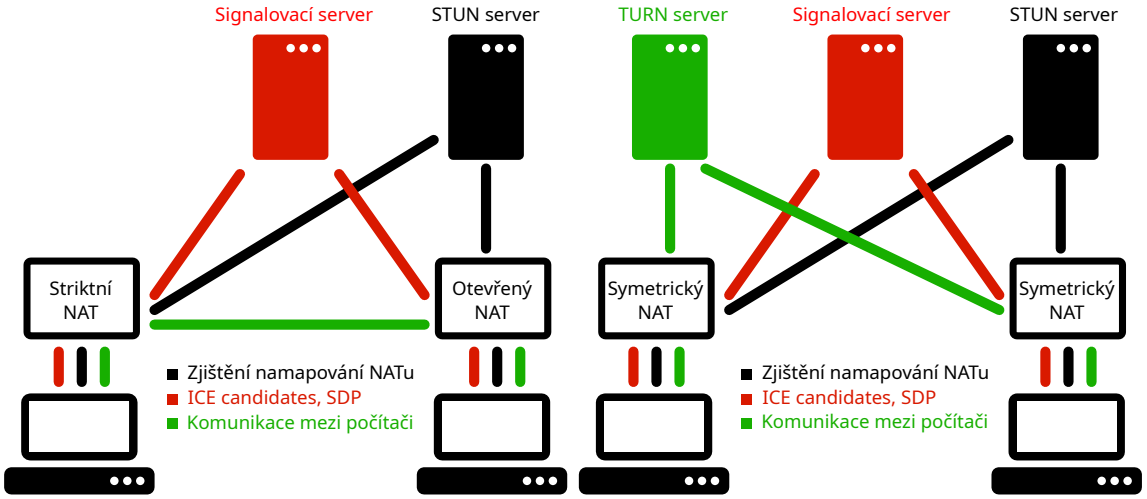
Jedná se o jediné standardizované aplikační rozhraní, které je dnes implementované v prohlížečích a zároveň umožňuje vytvářet přímé propojení mezi prohlížeči. Jako jediné umožňuje používat pro přenos ztrátovou komunikaci.

WebRTC poskytuje rozhraní, které dokáže přenášet real-time video a audio přenosy. Pro přenos používá vhodné kodeky, které jsou schopny se vypořádat se ztrátou paketů [4]. Umožňuje vytvářet i datové kanály, u kterých si může programátor specifikovat, zda chce přenášet data ztrátově, bezztrátově, uspořádaně nebo neuspořádaně. K multiplexování stremů jsou použity aplikační protokoly Stream Control Transmission Protocol[6] (SCTP)(datové kanály) a Secure Real-time Transport Protocol[7] (SRTP)(videa a audio přenosy). [4]

V jednom spojení může být více typů přenosů dat. Například na jednom přímém spojení mohou být dva datové kanály, jeden se ztrátovou komunikací a druhý s bezztrátovou komunikací [2].

[2] https://agar.io/
[3] https://diep.io/
[4] http://slither.io/
[5] https://krunker.io/
[6] https://datatracker.ietf.org/doc/html/rfc4960/
[7] https://datatracker.ietf.org/doc/html/rfc3711/

Komunikace je plně šifrovaná přes protokol Datagram Transport Layer Security[8] (DTLS) [4].



(a) Přímé propojení (b) Propojení přes TURN server

Obrázek 2.5: Zařízení použité ve WebRTC propojení

K vytvoření přímého spojení používá protokol Interactive Connectivity Establishment[9] (ICE). Pro vytvoření přímého spojení je nutné vyměnit ICE candidates mezi klienty přes signalovací server. Přes tento server se navíc posílají i informace o použitých kodecích a kanálech v Session Description Protokolu[10] (SDP). [4]

ICE candidates obsahují informace o připojení, přesněji metody, pomocí kterých se mohou propojit mezi sebou. Metody bývají seřazeny od nejlepších po nejhorší. [14]

Typicky mezi metodami propojení bývají uvedeny: propojení na lokální síti, propojení skrze NAT a propojení přes TURN[11] (proxy) server, pokud selže UDP děrování[12] za pomocí STUN[13] serveru.

1. **Propojení na místí síťi**
   U metody propojení na místní síti bývá uveden multicast DNS[14] (mDNS) záznam a udp port pro propojení. Počítače se snaží propojit na základě těchto mDNS záznamů mezi sebou. Pokud se nepovede mDNS záznam převést na IP adresu, nebo jestliže počítače přímo na sebe v lokální síti nevidí, spojení touto metodou selže.

   Dříve v této metodě byla uvedena lokální IP adresa, ale jelikož se zneužívala pro analytické účely, byla změněna za mDNS záznam [3, 18]. Na některých prohlížečích každopádně ještě není používaný mDNS záznam. Na základě mého experimentování jsem například zjistil, že není použit v prohlížeči Chrome na Androidu v době psaní této práce (květen 2022). Pokud prohlížeče nepoužívají mDNS záznam, propojení selže pokud počítače nejsou schopny se přímo připojit navzájem přes lokální adresy.

[8] https://datatracker.ietf.org/doc/html/rfc4347
[9] https://datatracker.ietf.org/doc/html/rfc8445
[10] https://datatracker.ietf.org/doc/html/rfc4566
[11] https://datatracker.ietf.org/doc/html/rfc5766
[12] https://datatracker.ietf.org/doc/html/rfc5128#section-3.3
[13] https://datatracker.ietf.org/doc/html/rfc5389
[14] https://datatracker.ietf.org/doc/html/rfc6762

2. **Propojení skrze NATy**
   K vytvoření ICE candinate pro propojení skrze NATy musí prohlížeč kontaktovat STUN server, na kterém zjistí svoji veřejnou IP adresu a namapovaný port, na který se jeho protějšek poté pokusí připojit. Druhé zařízení provede stejnou operaci a navzájem si vymění tyto kandidáty s adresami a porty. [4, 21] Jak je vidět na obrázku 2.5a. Tento typ propojení selže v případě, kdy jsou počítače za symetrickým NATem. Symetrický NAT neumožní počítačům využít stejné mapování, které zjistily při komunikaci se STUN serverem [21]. WebRTC se nepokouší predikovat nové mapování. Pokud by WebRTC provádělo predikci, bylo by schopné se u některých symetrických NATů touto metodou propojit.

   Pro vytvoření mapování na NATu jdou použít i speciální protokoly jako je například Universal Plug and Play[15] (UPnP), nebo NAT Port Mapping Protocol[16] (NAT-PMP), přes které jde přímo provést port forwarding, ale opět nejsou podporovány bohužel ze strany WebRTC [2].

3. **TURN server**
   Pokud vše selže, je nakonec použit TURN server, který pouze přeposílá data mezi klienty. Na obrázku 2.5b je znázorněno přeposílání zpráv přes TURN server.

Vývojář pro výměnu ICE candidates může využít libovolné protokoly (například: SIP, Jingle, ISUP) a aplikační rozhraní [4]. Nejčastěji jsem viděl využitý WebSocket, který je díky svým vlastnostem i nejvhodnější k tomuto účelu.

WebRTC upřednostňuje UDP protokol pro propojení, pokud nelze vytvořit propojení pře UDP protokol může vytvořit propojení přes TCP protokol, bohužel ne všechny prohlížeče tuto funkci podporují. [14]

Pro vytváření přímého spojení je každopádně lepší používat UDP protokol, vytvoření spojení přes UDP je podporováno na více zařízeních [19].

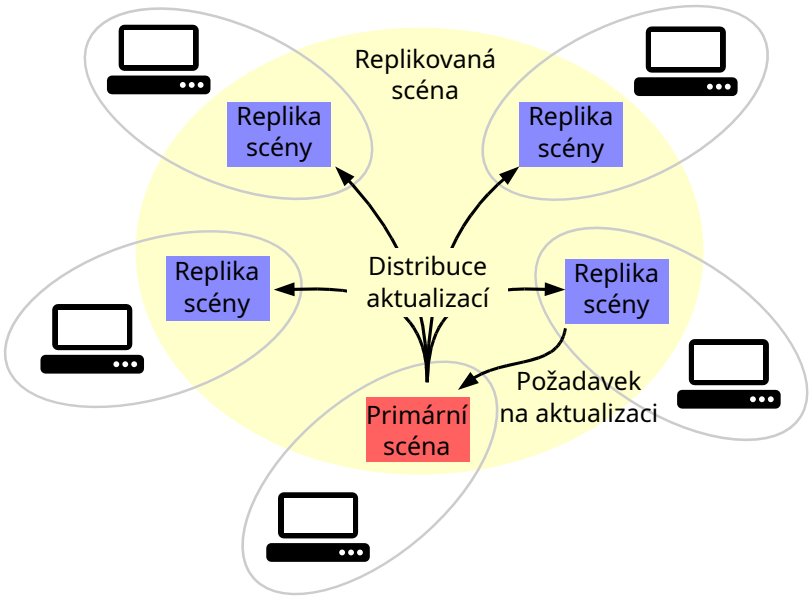
Toto rozhraní se dnes převážně používá v aplikacích pro videohovory. Najdeme ho například v aplikacích Skype[17] a Discord[18].

2.4 Konfigurace kolaborativní scény

Centalized primaries

V této konfiguraci je primární scéna na hlavním zařízení (serveru) a všichni ostatní (klienti) mají její repliku. Pokud chtějí scénu aktualizovat, musí požádat server o aktualizaci. Server může aktualizaci potvrdit, nebo zamítnout. Po provedení aktualizace, jsou o aktualizaci informovaní všichni klienti. [17]

[15] https://datatracker.ietf.org/doc/html/rfc6970
[16] https://datatracker.ietf.org/doc/html/rfc6886
[17] https://www.skype.com/
[18] https://discord.com/

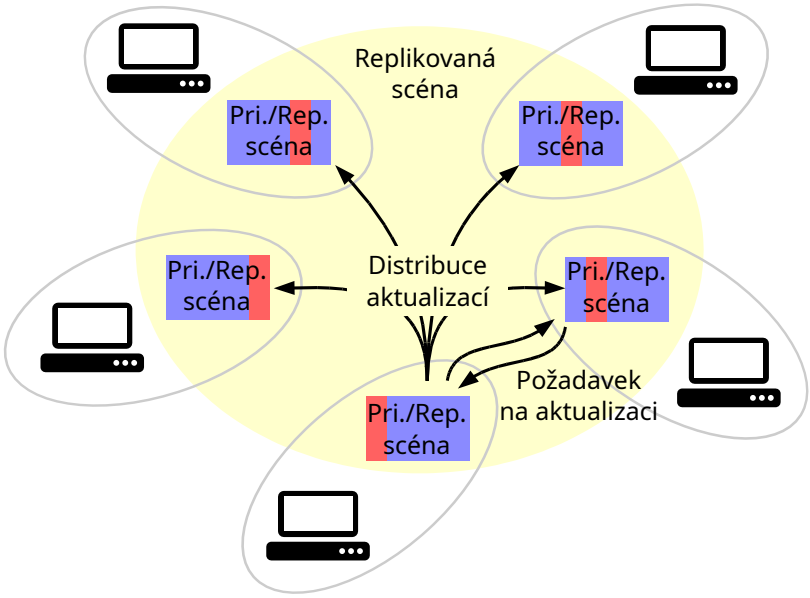


Obrázek 2.6: Centalized primaries [17]

Tato konfigurace má velmi silnou validaci a velmi jednoduše se navrhuje. Její nevýhodou je závislost na jednom zařízení (serveru). Pokud server spadne, celý systém se stává nefunkčním. [17]

Distributed primaries

Je velmi podobný předchozí konfiguraci. Na rozdíl od předchozí, účastníci vlastní určitou část scény. Pokud někdo chce provést aktualizaci scény, kterou nevlastní, musí požádat o aktualizaci konkrétní zařízení, vlastnící danou část scény. Dané zařízení buď aktualizaci přijme a odešle ji na ostatní zařízení, nebo ji zamítne. [17]

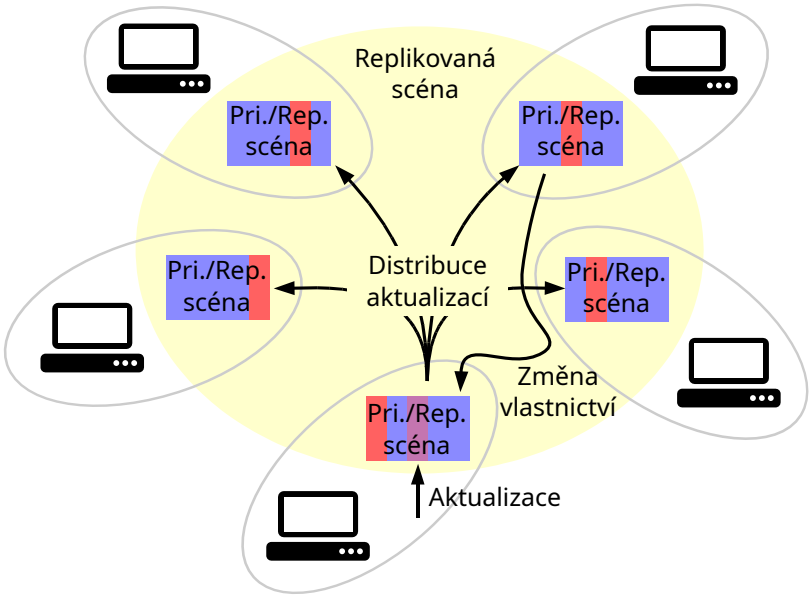


Obrázek 2.7: Distributed primaries [17]

Toto řešení sebou přináší horší konzistenci dat, ale zvyšuje škálovatelnost scény. Jedna velká scéna je rozložená mezi zařízeními a každé má na starosti svoji přidělenou část. [17]

Data ownership

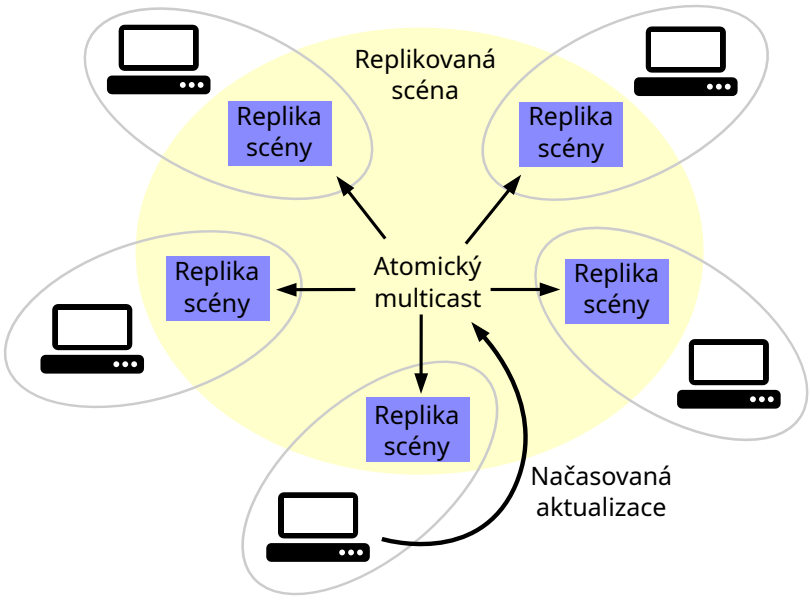
Je velmi podobný Distributed primaries. Umožňuje navíc změnu vlastnictví části scény. Zařízení může provádět zápis jen do častí scény, kterou vlastní. Pokud chce nějaký účastník zapsat do části scény, kterou nevlastní, musí požádat o změnu vlastnictví. Poté co část scény začne vlastnit, může do ní zapsat nová data. Následně odešle ostatním informaci o aktualizaci scény. Bez vlastnictví mohou zařízení z dané části scény jen číst a nemají garanci nejnovějšího stavu scény. [17]



Obrázek 2.8: Data ownership [17]

Aktivní replikace

Aktivní replikace jsou založeny na principu determinizmu. Každý účastník má repliku scény, a pokud každý účastník dostane stejné vstupy, vytvoří stejné výstupy. Pokud nějaké zařízení chce provést aktualizaci scény, odešle tuto aktualizaci na všechna zařízení pomocí atomického multicastu. Ostatní zařízení nepotvrzují změnu kolaborativní scény. Díky tomu se latence oproti předchozím modelům dvojnásobně sníží. [17]



Obrázek 2.9: Aktivní replikace [17]

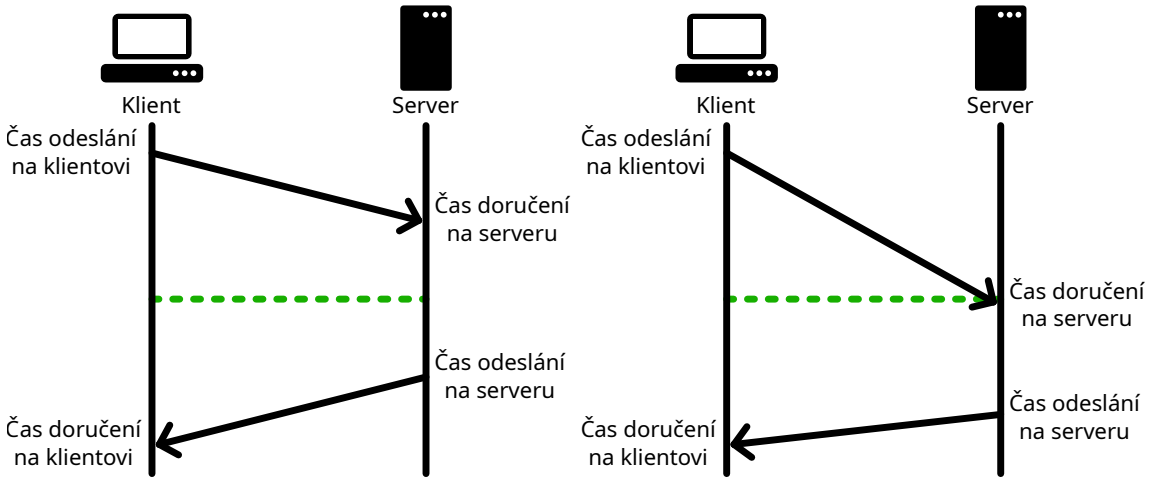
2.5 Systémové hodiny a monotónní hodiny

Systémové hodiny reprezentují čas na lokálním počítači. Jsou ovlivňovány synchronizací času operačního systému a mohou se posouvat zpět v čase [6].

Pokud jsou hodiny monotónní, plynou od určitého okamžiku neustále dopředu a nemohou se posouvat zpět v čase. Rychlost běhu těchto hodin může být měněna NTP klientem. [6] Například pokud uživatel přenastaví systémové hodiny zpět v čase, monotónní hodiny se neposunou zpět v čase.

2.6 Synchronizace času přes paketovou síť

NTP



(a) Ideální odezva sítě (b) Reálná odezva sítě

Obrázek 2.10: NTP synchronizace času

Při synchronizaci času se serverem klient uvede ve svém dotazu svůj aktuální čas hodin (t_1). Server si zaznamená čas doručení dotazu podle svých (referenčních) hodin (t_2). Odešle zpátky klientovi čas odeslání jeho dotazu podle jeho hodin (t_1), čas doručení dotazu podle hodin serveru (t_2) a čas odeslání odpovědi na klienta podle času serveru (t_3). Klient si zaznamená čas doručení podle svých hodin (t_4). [6]

Rovnice 2.1 popisuje čas strávený přenosem paketů po síti. Rovnice byla převzata z [6].

$$\delta = (t_4 - t_1) - (t_3 - t_2) \tag{2.1}$$

V algoritmu se počítá se symetrickou cestou (doba přenosu dotazu a odpovědi je stejná). V reálné síti se tato doba může lišit a způsobuje nepřesnost synchronizace. [6] Jak je znázorněno na obrázku 2.10.

Klient následně vypočítá, o kolik musí své hodiny posunout, aby byly synchronizovány k časovému serveru [6].

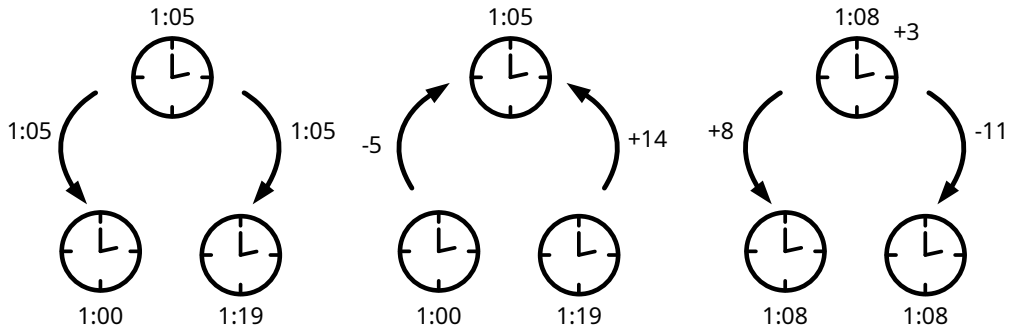
Rovnice 2.2 popisuje posun hodin, tak aby byly synchronizovány k serveru. Rovnice byla převzata z [6].

$$\theta = t_3 + \frac{\delta}{2} - t_4 \tag{2.2}$$

Pokud je odchylka k serveru malá NTP klient zrychlí nebo zpomalí běh hodin (slewing), je-li odchylka větší, proběhne přímo nastavení hodin (stepping) a nakonec když je odchylka příliš veliká (výchozí hodnota více jak 15 minut) klient předpokládá, že se něco pokazilo a odmítne změnit čas svých hodin [6].

2.6.1 Berkeley algoritmus

Při synchronizaci hodin se zeptá zařízení, které synchronizuje hodiny, jaký je čas na ostatních zařízeních. Na základě těchto informací vypočítá průměrnou hodnotu času všech hodin a odešle informaci každému zařízení, o kolik musí, posunou své hodiny, aby na všech zařízeních byl stejný čas. Tento algoritmus je vhodný pro systémy, které nemají referenční hodiny. [20] Grafické znázornění synchronizace času je uvedeno na obrázku 2.11.



(a) Dotaz jaký je čas na ostatních zařízeních

(b) Odpověď na dotaz

(c) Posunutí času na zařízeních

Obrázek 2.11: Synchronizace hodin pomocí Berkeley algoritmu

2.7 Iluze nulové latence

Standardně klient čeká na potvrzení změny, kterou provedl. To narušuje plynulost uživatelského rozhraní. Tady nad touto latencí jde vytvořit iluze nulové latence tím, že se uživateli okamžitě zobrazí změna a pokud změna není schválena z druhé strany, je zpětně stav lokální scény navrácen. Každopádně návrat se děje jen velmi výjimečně v typických aplikacích. [17]

2.8 Konzistence kolaborativní scény

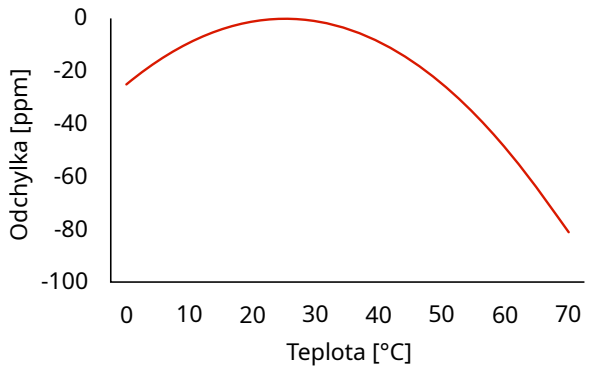
Kongruentní generátor

Kongruentní generátor deterministicky generuje pseudonáhodná čísla na základě semínka. Za použití stejného semínka bude generovat stejná čísla. Posloupnost vygenerovaných čísel je konečná a po určité době se začne opakovat. [16]

Chyby desetinných čísel

Výpočty na desetinné řádce mohou vycházet na různých architekturách jinak. Popřípadě mohou vycházet i rozdílně pokud jsou použity jiné překladače. Další možností různých výsledků může být použití odlišných typů instrukcí. Tady tato nejednota ve výsledcích narušuje determinismus mezi zařízeními. [17]

Odchylka hodin



Obrázek 2.12: Odchýlení od referenční frekvence [6]

V dnešních počítačích se pro chod hodin používají převážně křemíkové oscilátory. Odchylka křemíkových oscilátorů různě kolísá podle okolní teploty, kvůli této vlastnosti je potřeba občas na počítačích synchronizovat čas k přesnějším hodinám. Většina počítačů má odchylku hodin do 50 ppm. [6]

Kapitola 3

Návrh

Text v této kapitole popisuje návrh aplikace, jež demonstruje řešení, které dokáže co nejvíce snížit latenci při hraní hry a nemá používat dnes typicky používanou architekturu klient-server s aplikačním rozhraním WebSocket, který je dnes převážně využívaný u populárních webových her, jelikož zvyšuje latenci.

3.1 Popis hry

Hráč vytvářející hru nastaví dobu trvání hry (výchozí hodnota 15 minut). Pozve své spoluhráče (jednu hru mohou hrát 2-4 hráči). Na začátku se vygeneruje mapa, na které bude hra probíhat. Každý hráč na začátku dostane přidělené stejné množství peněz. Postupem času budou na planetě přibývat firmy, které budou na prodej a hráči si je budou moci koupit. Každá takováto firma bude mít různé výnosy, které budou kolísat, výnosy firem budou ovlivňovány úrovní firmy. Úrovně firem půjdou dokupovat. Hráč uvidí cenu firmy, rozsah v jakém zisk firmy kolísá, zda je firma na prodej, nebo patří protihráči.

Vyhrává hráč, který bude mít na konci hry nejvíce peněz ze všech hráčů.

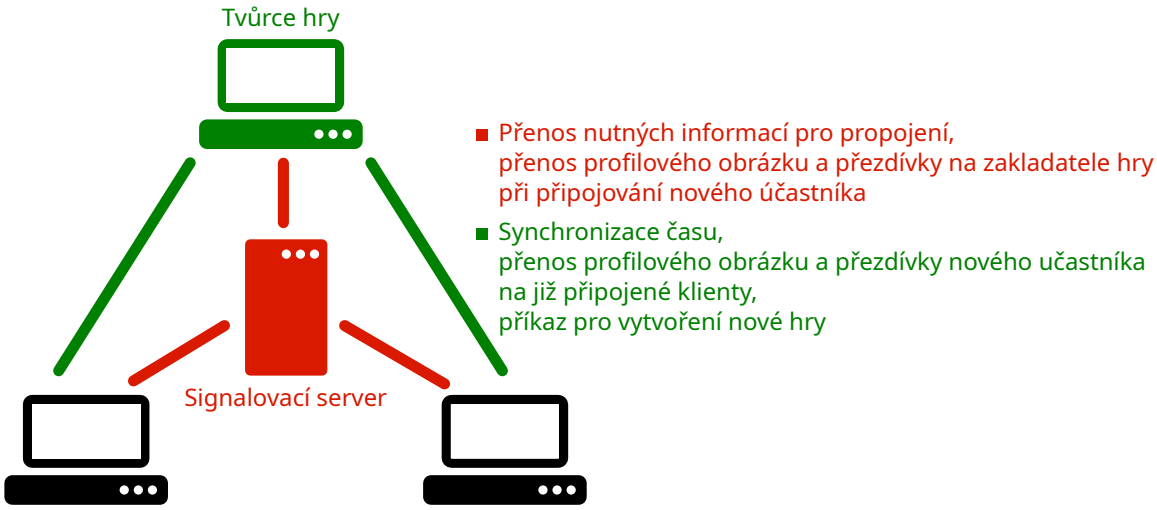
3.2 Síťová architektura

Při návrhu aplikace jsem se rozhodl použít dva hlavní typy síťových architektur, které se během běhu aplikace mění. Vyjímkou je synchronizace času, která svoji síťovou architekturu nemění a používá neustále architekturu klient server. Při návrhu jsem se snažil co nejvíce redukovat latenci při probíhající hře.

První architektura, která je použita při vytváření hry a synchronizaci herních hodin je typu klient server. Tady tuto architekturu jsem se rozhodl použít při vytváření hry z důvodu, že v této době ještě není plně jisté, zda všichni účastníci se zúčastní nové hry. Vytvoření a udržování přímého spojení navíc přináší další režie. Během vytváření přímého spojení musí být přenesena data přes STUN server, signalovací server a musí být vyměněna počáteční data mezi peery. Poté co je vytvořeno přímé spojení, WebRTC musí udržovat mapování na NATu, nebo firewallu pomocí keep-alive paketů, aby nedošlo k ukončení přímého spojení, pokud se delší dobu nic nepřenáší. Na začátku hry by se stejně nic na přímo posílat nemohlo, kromě přezdívky hráče a profilového obrázku. Takže by jen vznikaly zbytečné režie navíc a ještě by to způsobilo vyšší latenci při přenosu těchto dvou údajů, jelikož vytvoření přímého spojení je velmi náročná a zdlouhavá operace. Jak je popsáno v sekci 2.3.

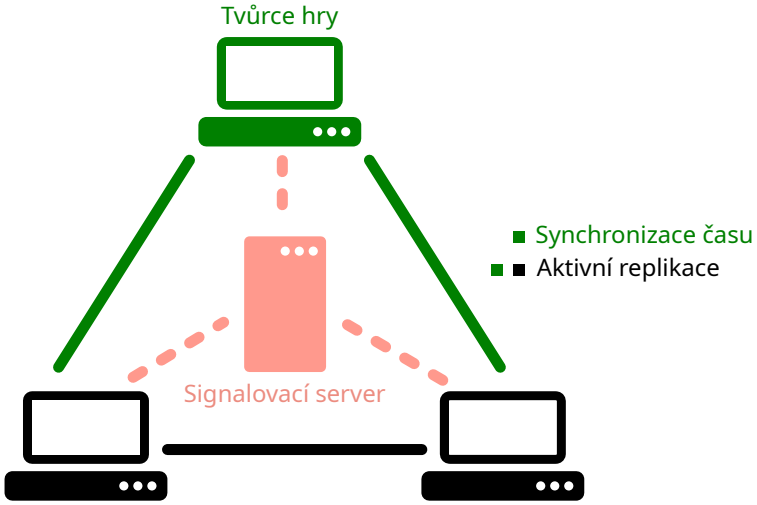
Z této použité architektury plyne použití konfigurace kolaborativní scény na centalized primaries, která je k tomuto účelu nejlepší. Zakladatel hry má primární scénu a všichni ostatní hráči, kteří se připojují, mají repliku primární scény.

Při připojování nového účastníka do hry jsou jeho informace o profilu (profilový obrázek a přezdívka) přímo přenášeny přes signalovací server na zakladatele hry, aby nemusel čekat na vytvoření přímého spojení a mohl odeslat tyto informace rovnou k již připojeným účastníkům. Profilový obrázek je přenášen v png formátu s rozlišením 128x128 pixelů. Po vytvoření přímého spojení na zakladatele hry, jsou odeslány informace o připojených hráčích na nově připojeného účastníka. Zároveň po vytvoření spojení začne okamžitě probíhat synchronizace herních hodin k zakladateli hry v intervalu 1 sekundy. Tato architektura je zjednodušeně znázorněná na obrázku 3.1.



Obrázek 3.1: Architektura sítě při vytváření nové hry

Jakmile jsou všichni hráči připraveni a zakladatel hry nastaví dobu trvání hry 1 - 60 minut. Je odeslána zpráva od zakladatele hry s parametry nutnými pro spuštění nové hry. Na všech zařízeních se spustí odpočet do začátku hry. Během tohoto odpočtu je vytvořeno přímé spojení se všemi účastníky, kteří následně komunikují na přímo mezi sebou jak je znázorněno na obrázku 3.2, aby byla snížena co nejvíce latence sítě.

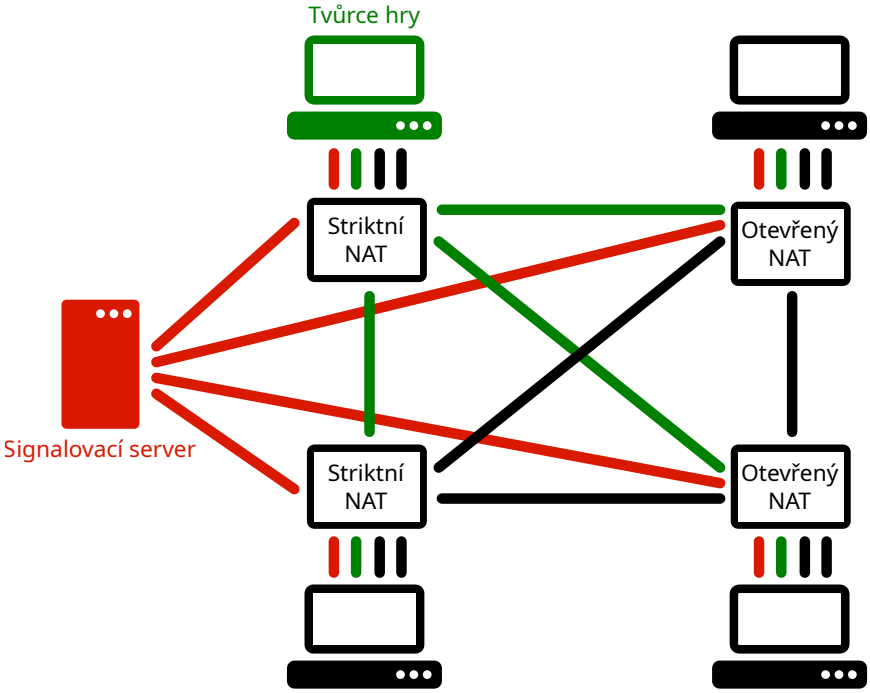


Obrázek 3.2: Architektura sítě při probíhající hře

Pro přímou komunikaci mezi zařízeními jsem se rozhodl využít aktivní replikace, které se k tomuto účelu nejvíce hodí. Hra nemá žádnou složitou scénu, aby bylo třeba použít konfiguraci distributed primaries a zároveň aktivní replikace umožňují dobře potlačovat latenci sítě. Zprávy nemusí chodit přes žádné centrální zařízení, které by v dané části scény schvalovalo změny. Zprávy, díky tomu, mohou vždy chodit nejrychlejší vytvořenou cestou na síti. Použití centalized primaries by zase mohlo zbytečně přidávat latenci navíc (záleží na poloze serveru na síti).

Použití konfigurace data ownership by bylo taky celkem nevhodné. Při koupení firmy by zařízení muselo někdy žádat o změnu vlastnictví dané části scény a díky tomu by se zvyšovala latence při kupovaní firem a hra by působila méně plynule.

Na obrázku 3.3 je zobrazena ukázková detailnější síť, jak spojení mezi zařízeními může vypadat během probíhající hry.



Obrázek 3.3: Detailní síť

Synchronizace času po celou dobu hry funguje na architektuře klient server, stejně jako při připojování do nové hry.

Chat se přizpůsobuje stavu propojení klientů. Při vytváření hry se zprávy odesílají přes architekturu klient server, kde server je zakladatel hry. Během probíhající hry, kde je vytvořené mezi všemi účastníky přímé spojení. Jsou přímo odesílány zprávy na každého účastníka na přímo.

Pro přizpůsobování architektury komunikace chatu jsem se rozhodl z důvodu, že na začátku hry nemá smysl vytvářet přímé propojení jen kvůli snížení latence chatu, který z převážné většiny při zakládání hry není využívaný. Při již vytvořeném propojení všech účastníků je zase vzhledem k latenci, zprávy lepší posílat na přímo.

Spojení se ukončí na konci hry se zpožděním několika sekund, je to z důvodu čekání na možné opožděné zprávy, které mohou ještě změnit výsledky hry.

Všechna vytvořená spojení mezi klienty používají bezztrátovou a uspořádanou komunikaci, která je pro tuto aplikaci vhodnější. Jednoznačně by pro aplikaci nebylo vhodné používat ztrátovou komunikaci. Rozhodně by žádný hráč nechtěl, aby například informace, že koupil firmu, se na síti ztratila. Uspořádaná komunikace na druhou stranu usnadňuje implementaci, pokud by nebyla použita. Musel bych navíc řešit situace, jako například informace o změně úrovně firmy došla dříve, než informace o koupě firmy. Takže bych tuto informaci musel podržet někde v paměti a čekat, než přijde informace o koupě firmy, jelikož by v tu chvíli nešlo zjistit, komu firma doopravdy patří a stejně by aplikace mohla výsledek zobrazit po přijetí obou zpráv.

3.3 Připojení do hry

Při navrhování způsobu propojení hráčů mezi sebou mě napadly dvě možnosti. První metoda, která mě napadla, bylo seznámit jednotlivé hráče na základě URL odkazu, který se

dá jednoduše komukoliv odeslat přes jakoukoliv chatovací aplikaci. Další možnost, kterou jsem zvažoval, byla mít čekací místnost pro hráče, kde by čekali než by se neobjevili další hráči, kteří budou chtít hrát. Hráče by poté seznámil navzájem server, který by spravoval čekací místnosti.

Druhá možnost by vyžadovala nutnost navíc hostovat server, který by byl s velkou pravděpodobností funkční jen krátkou dobu. Zároveň jsem chtěl, aby byl schopný každý spustit aplikaci bez nutnosti nastavovat jakýkoliv server, nebo zprovozňovat webový server i podobě co mnou provozovaný server přestane fungovat. Na základě těchto požadavků jsem se rozhodl použít odkaz, ze kterého čte potřebné informace rovnou JavaScript. Každopádně varianta s odkazem pořád potřebuje signalovací server a STUN server. Jsou to minimální požadavky na vytvoření přímého spojení za použití aplikačního rozhraní WebRTC.

Připojení do hry jsem tedy řešil přes URL odkaz, který se zobrazuje v aplikačním rozhraní po založení nové hry. Tento odkaz obsahuje UUID, které identifikuje na signalovacím serveru zakladatele hry.

3.4 Synchronizace času

Synchronizaci času jsem se rozhodl dělat velmi podobně, jak ji dělá protokol NTP. Berkeley algoritmus jsem nevyužil z důvodu, že bych musel synchronizovat minimálně jednou hodiny před začátkem hry mezi všemi účastníky a až po synchronizaci hodin by aplikace mohla vytvořit informace o nové hře, které by následně odeslala ostatním.

Během synchronizace času jsem se rozhodl počítat s odchylkou hodin 50 ppm na základě sekce 2.8. Každopádně většina zařízení má mnohem nižší odchylku podle mých experimentů a navíc pokud je na zařízení NTP klient, tak ještě provádí korekci rychlosti běhu hodin. Každopádně nemůžu počítat, že na zařízeních bude nainstalován. Dále velmi záleží na tom, jak je zařízení vyrobeno. Jak je vidět v sekci 2.8 odchylka hodinového krystalu velmi záleží na okolní teplotě. Pokud je krystal umístěn na nesprávném místě, kde bývají vyšší teploty, jeho odchylka se zvyšuje.

Synchronizaci času jsem se rozhodl dělat v intervalech 1 sekundy. Interval jsem záměrně zvolil kratší, aby nebylo nutné odesílat keep-alive pakety. Po provedení každé synchronizace hodin je porovnána přesnost synchronizace s aktuální odchylkou hodin k referenčním. Pokud hodiny mají vyšší odchylku než synchronizace, jsou přenastaveny. Navržený algoritmus navíc každou sekundu zvyšuje odchylku lokálních herních hodin o 0,05 ms (50 ppm).

3.5 Iluze nulové latence

Na začátku hry jsem se rozhodl, že na zařízení které vytváří hru bude vygenerováno semínko pro deterministickou simulaci. Na základě tohoto vygenerovaného semínka a synchronizovaných herních hodin probíhá na všech zařízeních deterministická simulace.

Na začátku hry jsem se rozhodl na zařízení, které vytváří hru vygenerovat semínko pro deterministickou simulaci, která následně probíhá na všech zařízeních podle synchronizovaných herních hodin. Díky tomu se na všech zařízeních téměř ve stejný čas zobrazují nové firmy.

Pokud některý účastník provede jakoukoliv operaci s firmami, je okamžitě provedena a odeslána na ostatní peery. Pokud se zpětně zjistí, že například někdo koupil firmu dříve podle synchronizovaných herních hodin a měl na to dostatek peněz, nebo koupil firmu ve

stejnou dobu, ale připojil se do hry dříve. Je firmě zpětně změněno vlastnictví a odebrány všechny akce, které provedl neplatný vlastník firmy.

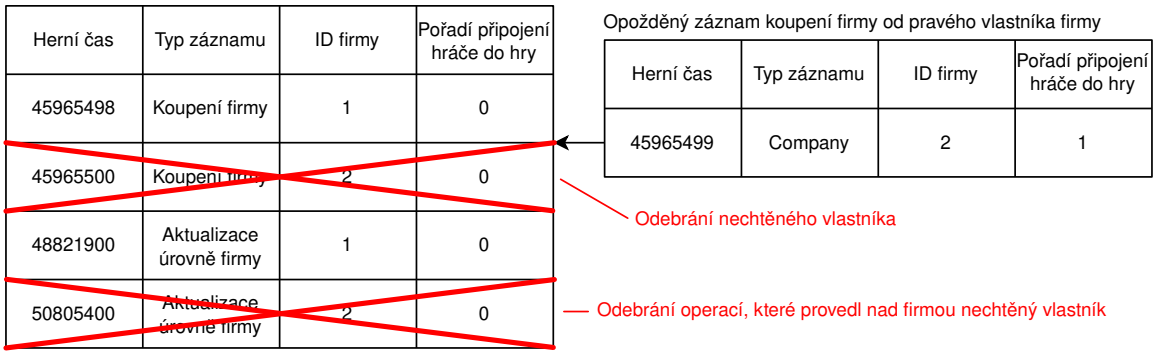
Iluzi nulové latence jsem se rozhodl řešit i v chatu. Pokud někdo odešle zprávu, je mu hned zobrazena v chatu. Pokud k němu přijde zpráva, která byla odeslána v dřívějším čase, je do chatu přidána dodatečně před zprávu, kterou odeslal. Pokud nastane případ, že zprávy od dvou účastníků byly odeslány ve stejný čas, seřadí se podle pořadí připojení do hry, stejně jako u operací s firmami.

3.6 Konzistentní model kolaborativní scény

Stav firem

Konzistenci firem jsem se rozhodl udržovat pomocí záznamů, které jsou uspořádány podle herního času. Jestliže přijdou zprávy od dvou účastníků ve stejnou dobu má přednost účastník, který se připojil do hry dříve. Pokud přijde zpráva, je zařazena podle času a pořadí připojení do hry mezi ostatní záznamy a jsou zkontrolovány všechny záznamy, které obsahují pozdější čas. Pokud některý záznam nesplňuje konzistentní model je odstraněn. Pokud je odstraněn záznam kupující firmu, jsou odstraněny všechny navazující záznamy, s informacemi co hráč s danou firmou prováděl.

Na obrázku 3.4 je znázorněno vložení nového záznamu a odstranění neplatných záznamů v případě opoždění zprávy koupení firmy skutečného vlastníka firmy.



Obrázek 3.4: Odstranění neplatných záznamů

Mapa

Mapu jsem se rozhodl generovat pomocí softwarového rendereru. Je vytvořena na základě semínka pro deterministickou simulaci spolu s pozicemi firem na mapě. Mapa musí být na všech zařízeních stejná spolu se stejnými pozicemi firem. Měla by se tedy vyvarovat problémům konzistence uvedených v sekci 2.8.

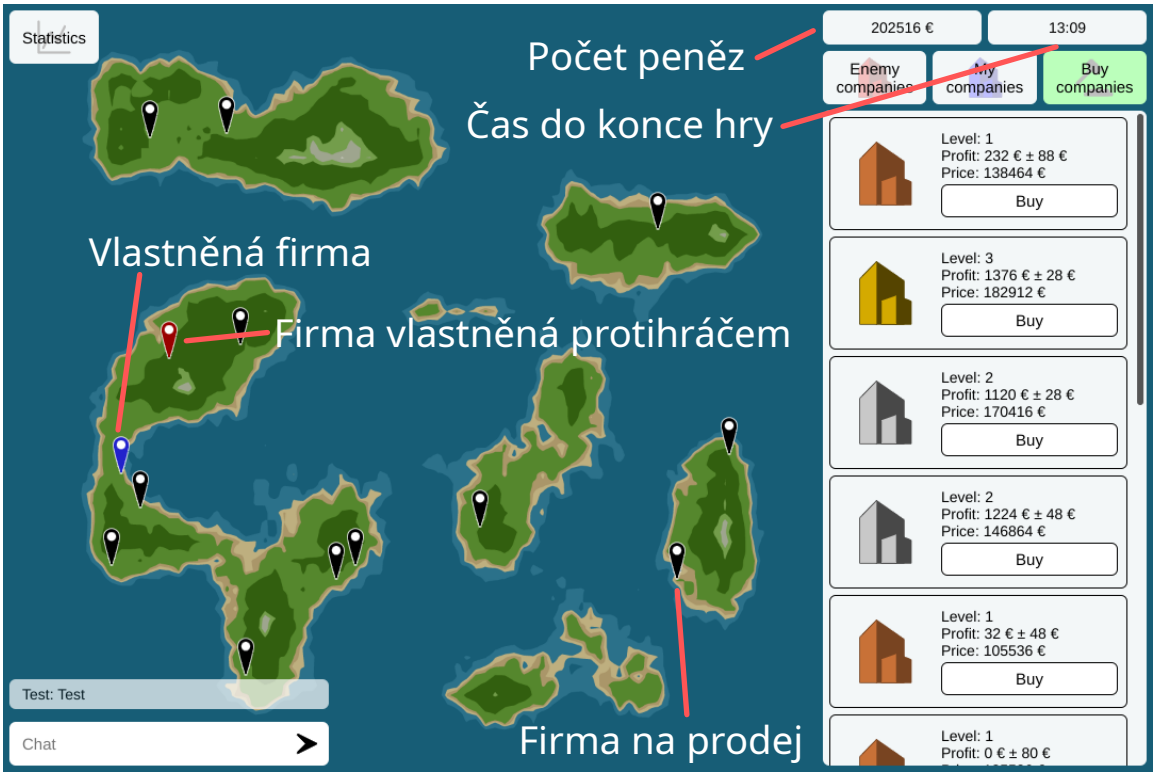
3.7 Uživatelské rozhraní a ovládání

Aplikace má dvě hlavní části uživatelského rozhraní. Jedno je přizpůsobeno na počítače a druhé na telefonní zařízení.

Hra se ovládá pomocí myši, nebo dotyku na dotykových zařízeních. Rozhraní se skládá ze tří hlavních částí mapy, ovládacího panelu obsahující tlačítka a chatu.

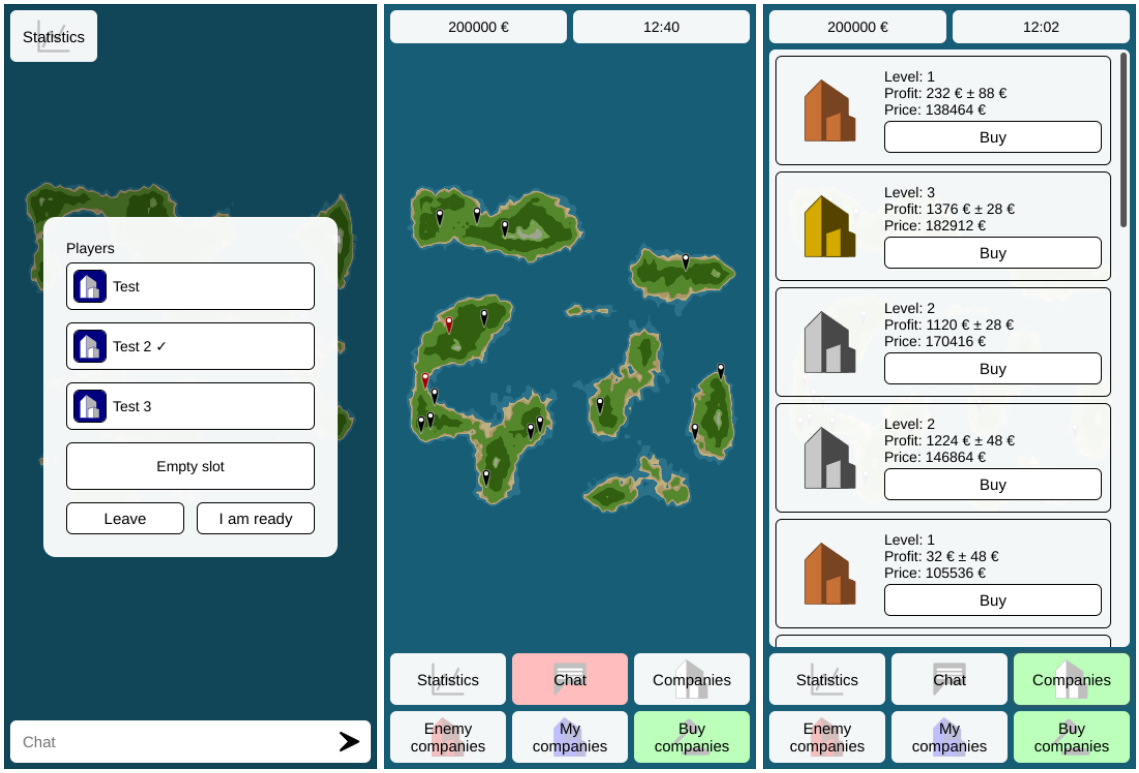
Rozložení aplikace na počítačích je zobrazeno na obrázku 3.5 a pro mobilní zařízení na obrázku 3.6.

(a) Založení nové hry



(b) Průběh hry

Obrázek 3.5: Počítačová verze



(a) Nová hra (b) Mapa (c) Firmy

Obrázek 3.6: Mobilní verze

Při kliknutí na jakoukoliv vyznačenou firmu na mapě, aplikace zobrazí informace o firmě v seznamu firem. Firmy vyznačené černě jsou na prodej. Červená barva označuje firmy protihráčů a modrá barva vyznačuje vlastněné firmy.

Kapitola 4

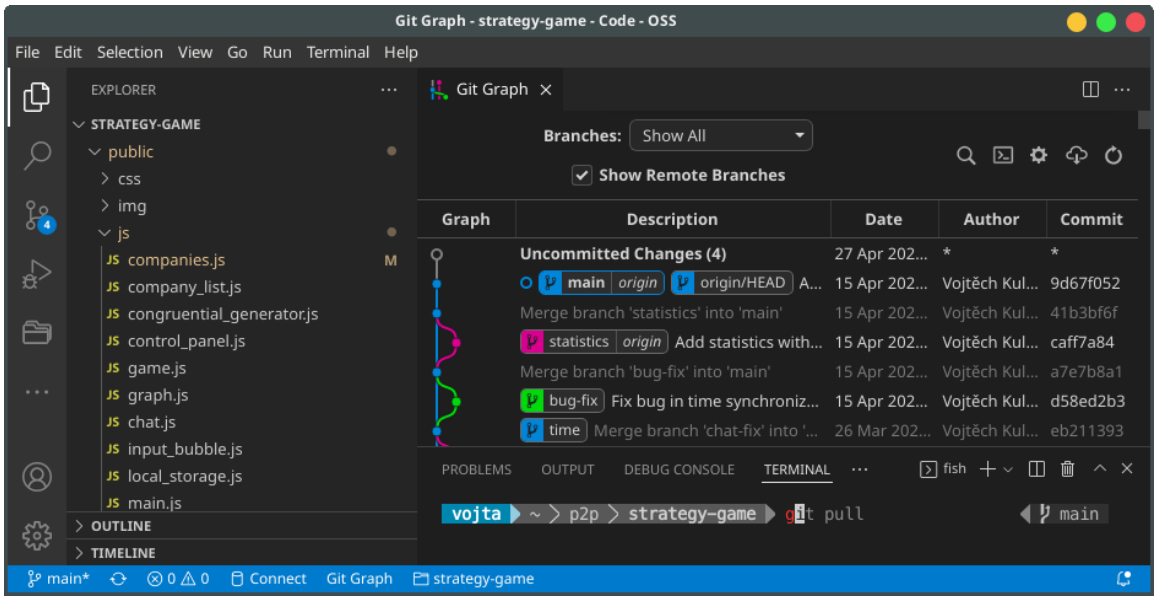
Implementace

V následující části, budou popsané detaily, jak probíhala implementace webové hry a jaké nastroje byly při implementaci použity.

4.1 Vývojové prostředí a repozitář

K zajištění lepší správy kódu jsem se rozhodl, vytvořil git repozitář na Gitlabu, který jsem otevřel komukoliv a kód zveřejnil pod licencí MIT. Gitlab jsem se rozhodl použít na základě dobrých předchozích zkušeností s touto službou. Repozitář je otevřený a k dispozici na adrese https://gitlab.com/kulisekvojtech/strategy-game pro všechny, které by kód aplikace zajímal.

Pro vývoj jsem použil Code - OSS[1] (open source distribuce Visual Studio Code) s rozšířeními EditorConfig[2], Git Graph[3] a Project Manager[4].



Obrázek 4.1: Code - OSS s rozšířeními

[1] https://github.com/microsoft/vscode/
[2] https://open-vsx.org/vscode/item?itemName=EditorConfig.EditorConfig
[3] https://open-vsx.org/vscode/item?itemName=mhutchie.git-graph
[4] https://open-vsx.org/vscode/item?itemName=alefragnani.project-manager

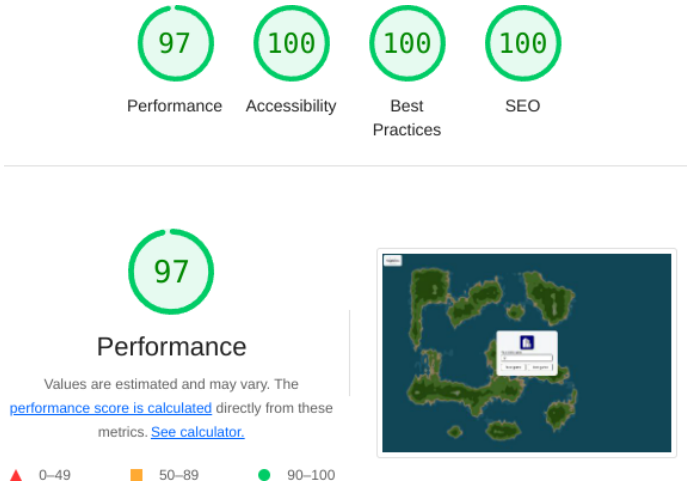
EditorConfig zajišťuje automatické nastavení editoru kódu, tak aby používal nastavení formátování z repozitáře. Mezi tato nastavení patří například typ odsazování, ukončení řádku a kódování souborů.

Git Graph umožňuje zobrazit graficky stav repozitáře a provádět na něm různé úpravy.

Project Manager slouží k jednoduchému přepínaní mezi více projekty.

4.2 Automatizované testování kvality aplikace

Během implementace jsem používal Lighthouse[5] k automatizovanému testování kvality aplikace, na základě jeho výsledku byla aplikace průběžně vylepšována. Na obrázku 4.2 jsou vidět výsledky z finální aplikace.



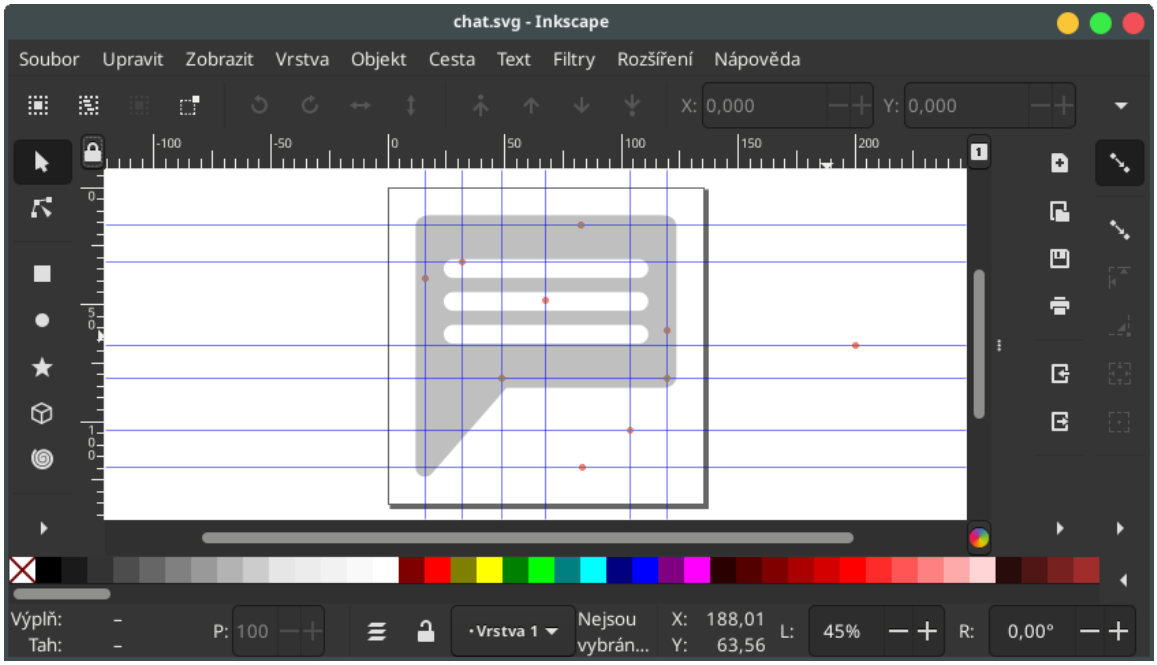
Obrázek 4.2: Část výsledků z automatických testů kvality Lighthouse

4.3 Obrázky

Veškeré obrázky ve hře jsem vytvořil sám s použití programu Inkspace. Všechny obrázky, kromě ikony hry, jsou vektorové a jejich zdrojový kód byl optimalizován optimalizátorem Scour[6], aby zabíraly co nejméně místa a co nejvíce redukovaly velikost aplikace.

Na obrázku 4.3 je vidět snímek obrazovky z aplikace Inkspace, při vytváření ikony otevření chatu pro mobilní verzi aplikace.

[5] https://developers.google.com/web/tools/lighthouse
[6] https://github.com/scour-project/scour



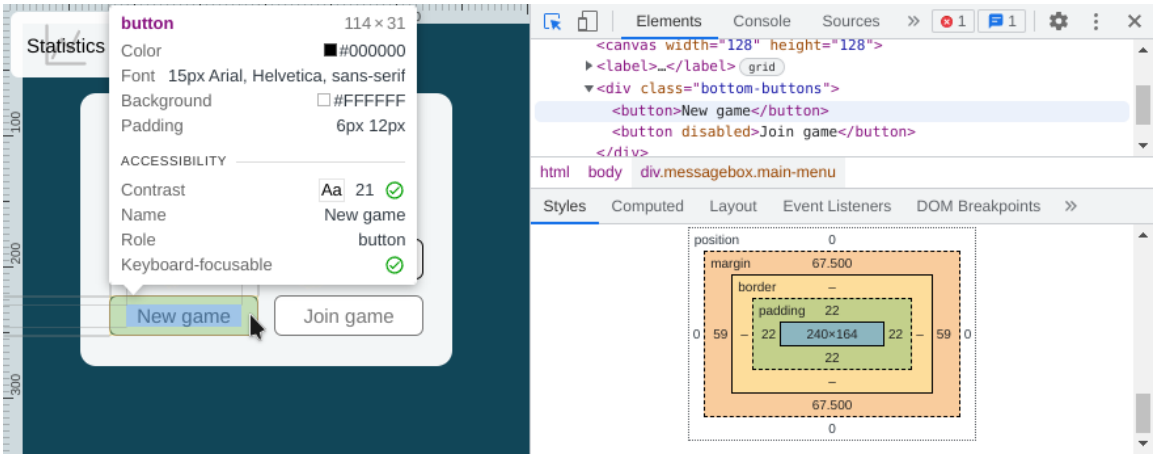
Obrázek 4.3: Inkspace s ikonou otevření chatu

Jakýkoliv obrázek prohlížeč stahuje ve chvíli, až ho potřebuje k zobrazení v uživatelském rozhraní. Například pokud uživatel nikdy neotevře statistiky, prohlížeč nikdy ze serveru hostující aplikaci nestáhne obrázek křížku k zavření statistik. Díky tomu se redukuje množství přenesených dat.

4.4 Uživatelské rozhraní

Uživatelské rozhraní jsem vytvořil pomocí kaskádových stylů, HTML5 a JavaScriptu, bez použití knihoven. Většina uživatelského rozhraní byla vytvořena za pomocí vývojářských nástrojů webového prohlížeče Chromium[7]. Nástroje jsou zobrazeny na obrázku 4.4.

[7] https://www.chromium.org/devtools/



Obrázek 4.4: Vytváření úvodní obrazovky pomocí vývojářských nástrojů aplikace Chromium

Během implementace jsem prováděl testy s uživateli a aplikační rozhraní jsem na jejich připomínky přizpůsoboval. Jako velmi užitečné vylepšení se ukázalo přidat na tlačítka text, který popisuje jejich funkčnost. Dříve měla tlačítka jen vektorové obrázky bez textu.

Knihovny jsem se rozhodl nevyužít z důvodu, že mám v těchto jazycích značnou praxi, programuji v nich dlouhou dobu, použití knihoven by mi zabralo více času z nutnosti procházet dokumentaci a celková aplikace by díky těmto knihovnám i více zabírala. Díky tomuto čistému zápisu jde i velmi jednoduše aplikaci graficky měnit, všechny styly jsou uloženy ve složce css s odpovídajícími názvy a není nutné při změnách vzhledu hledat jeho definici v JavaScriptu, popřípadě v HTML souboru.

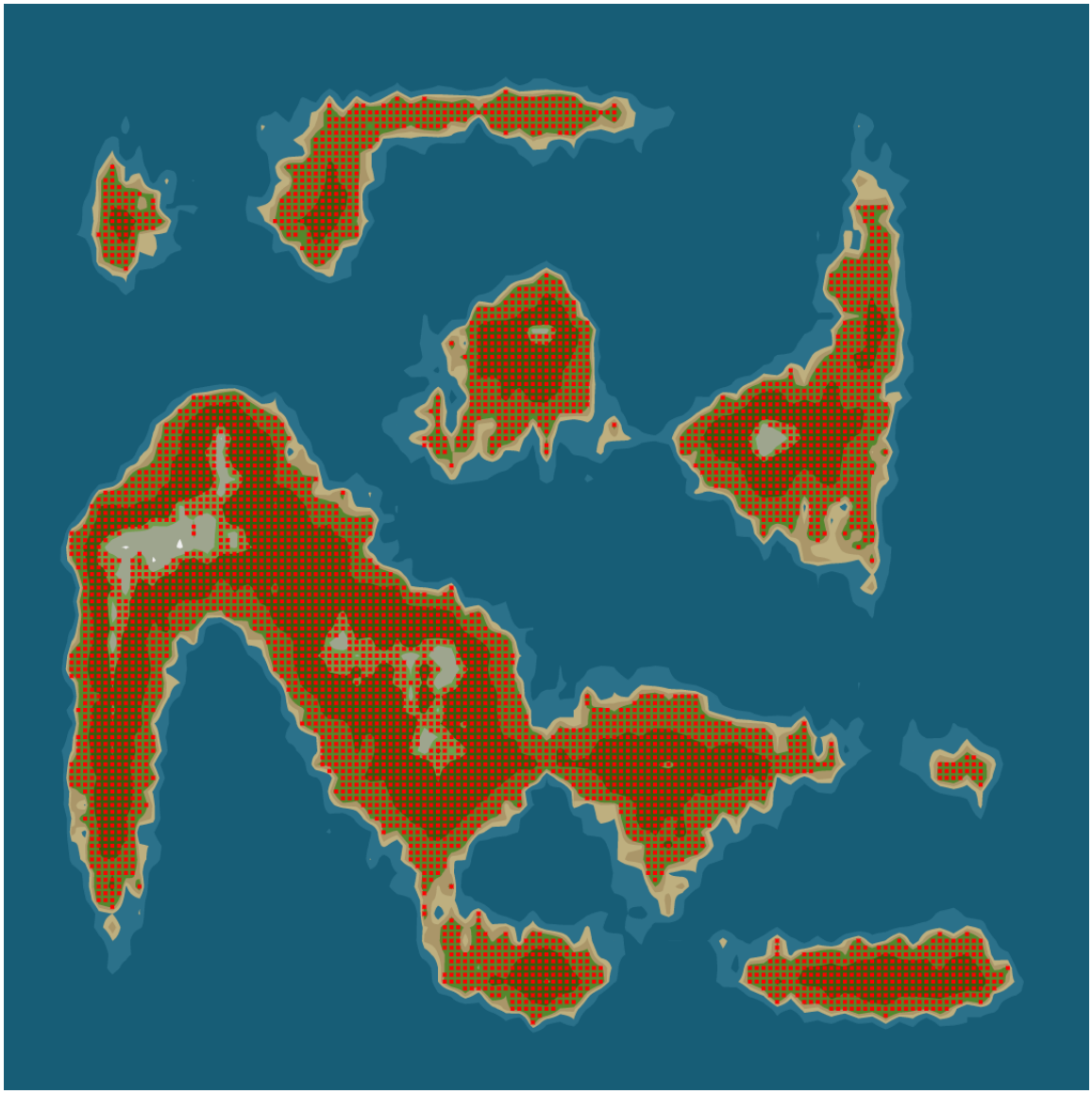
4.5 Mapa

Mapa je softwarově renderována, kromě pozic firem. Pozice firem jsou vytvořeny pomocí kaskádových stylů, HTML a JavaScriptu včetně animací, které jsou akcelerovány na grafické kartě. Nad každou vyznačenou pozicí firmy je neviditelný obdélník, který je před zprávami chatu a po kliknutí na něj, je zobrazena daná vyznačená firma v listu firem. Samotné obrázky vyznačující firmy jsou pod zprávami chatu. Rozhodl jsem se k této realizaci, abych umožnil uživatelům kliknout na firmu, která je na mapě pod zprávami chatu a jinak by na ni kliknout nešlo. Tyto neviditelné obdélníky jsou pod oknem statistik.

Mapa se generuje ve dvou hlavních částech. První část vůbec nepoužívá žádné operace dělení, aby nenastaly problémy s konzistencí uvedené v sekci 2.8. Druhá část používá dělení jen pro dokončení vykreslení mapy a to v interpolaci. Nakonec je provedeno prahování, jehož výsledkem je devět různých barev.

První část vytvoří dvourozměrnou mřížku, do které umístí čísla z kongruentního generátoru. V dalším kroku se vytvoří jemnější mřížka, ve které jsou vložené hodnoty z kongruentního generátoru ovlivněny předchozí mřížkou. Krok se ještě jednou naposled zopakuje.

Nad mřížkou je provedena interpolace pomocí logických posunů, která vytvoří velmi jemnou mřížku, z které jsou pomocí prahování získány povolené pozice firem. Pomocí mřížky povolených pozic a kongruentního generátoru jsou vygenerovány pozice jednotlivých firem na mapě. Data z mřížky zajistí, že se pozice firmy nevygeneruje v moři, na pláži, nebo na hoře. Na obrázku 4.5 je zobrazena tato mřížka.



Obrázek 4.5: Povolené pozice firem na mapě

Při generování pozic firem je ještě použit seznam již použitých míst. Další firmy se již na těchto pozicích a v jejich okolích nemůžou vytvářet, aby se jejich vyznačené body nepřekrývaly.

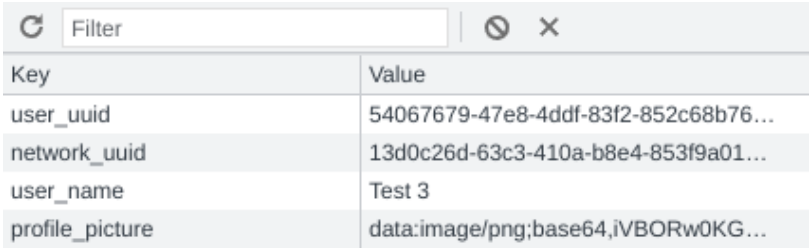
Poslední část vygeneruje mapu pomocí interpolace, za pomocí dělení a desetinných čísel. Výsledek interpolace je následně prahován na barvy.

4.6 Úložiště dat

Pro ukládání dat (posledního UUID, profilového jména a obrázku), jsem využil aplikační rozhraní localStorage[8]. Aplikační rozhraní localStorage umožňuje ukládat data do prohlížeče na dobu neurčitou [15].

Na obrázku 4.6 je vidět obsah úložiště aplikace.

[8] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage



Obrázek 4.6: Výpis obsahu localstorage ve vývojářských nástrojích Chromium

4.7 PeerJS

PeerJS[9] je open-source knihovna, která zjednodušuje práci s aplikačním rozhraní WebRTC. Knihovna je šířená pod MIT licencí.

Pro implementaci jsem se rozhodl použít tuto knihovnu, jelikož provozuje komunitní signalovací server. Knihovna navíc nad WebRTC řeší komunikaci se signalovacím serverem. Sama o osobě poskytuje jen rozhraní pro propojení dvou účastníků. Pokud programátor chce dělat sítě mezi více účastníky, musí řešit vytvoření všech propojení, seznámení všech účastníků atd.

Rozhodl jsem se využívat komunitní signalovací server, který knihovna nabízí. Tento server může každý používat bezplatně a díky tomu ho využívá více různých aplikací. Kvůli tomu, jsem se rozhodl, při připojování do hry odesílat přes signalovací server statický řetězec, který je následně při vytváření nového spojení kontrolován. Pokud chybí, nebo je jiný, aplikace ukončí nechtěné spojení s jinou aplikací.

4.8 Hodiny

Během implementace synchronizace hodin jsem se setkal s určitými omezeními, které přináší knihovna PeerJS. Nenašel jsem v dokumentaci žádný způsob, jak na jednom spojení vytvořit více datových kanálů s různými parametry. Což byl problém, jelikož synchronizaci času by bylo lepší dělat přes ztrátovou komunikaci.

Díky tomu jsem se rozhodoval mezi dvěma možnostmi. Buď posílat synchronizaci času přes již vytvořený uspořádaný a bezztrátový kanál, nebo vytvořit nové přímé spojení a způsobit další režie navíc při vytváření. Nakonec jsem se rozhodl použít již vytvořený datový kanál. Díky tomu jsou zase zbytečně znovu posílány ztracené pakety. Pokud se znovu posílají ztracené pakety, je nepřesnost synchronizace vyšší a algoritmus díky tomu ve většině případů synchronizaci odmítne.

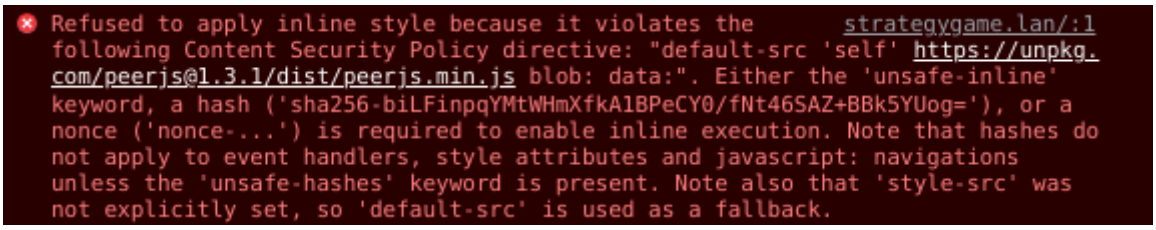
Pro vytváření herních hodin bylo použito aplikační rozhraní Performance. Toto rozhraní umožňuje přístup k monotónním hodinám. Čas je navrácen z tohoto rozhraní v datovém typu double [5]. Aplikace tuto hodnotu převádí na celočíselný datový typ, který reprezentuje mikrosekundy.

[9] https://peerjs.com/

4.9 Ochrana proti XSS

Pro zvýšení bezpečnosti jsem v souboru index.html definoval zdroje, se kterými aplikace může pracovat a z jakých zdrojů může spustit spustitelný kód. Při implementaci aplikace jsem navíc důkladně prověřoval části, ve kterých by mohlo dojít k tomuto typu útoku.

Na obrázku 4.7 je vidět výpis z konzole v prohlížeči Chromium pokud se do aplikace dostanou kaskádové styly z nepovolené lokace.



Obrázek 4.7: Zablokování kaskádových stylů

4.10 Použité třídy v aplikaci

Pro jednodušší rozšiřitelnost aplikace jsem se rozhodl použít objektově orientované programování. Aplikace je rozdělena do objektů uvedených v této sekci.

Třída companies

Vykonává veškeré operace s firmami a obsahuje stav jednotlivých firem. Umožňuje komukoliv kupovat firmy a měnit jejich stav.

Třída company_list

Během hry má tři instance. Jedna obsahuje všechny firmy, které jdou koupit. Druhá obsahuje všechny firmy, které hráč vlastní a poslední obsahuje všechny firmy, které vlastní protihráči. Všechny firmy vykresluje v grafickém rozhraní do listů.

Třída congruential_generator

Obsahuje implementaci kongruentního generátoru. Na základě semínka generuje deterministickou posloupnost pseudonáhodných čísel. Posloupnost pseudonáhodných čísel z této třídy je použita jako zdroj pro deterministickou simulaci, která probíhá během hry.

Třída control_panel

Implementuje panel, který slouží k ovládání hry. Zobrazuje v uživatelském rozhraní čas do konce hry, množství peněz. Umožňuje přepínat mezi listy s organizacemi.

Třída game

Řídí odpočet do začátku hry, průběh hry a zobrazení výsledků hry.

Třída graph

Vytváří graf ve vektorové grafice na základě vstupních dat. Vygenerovaná vektorová grafika je popsána ve formátu SVG a následně je vykreslena pomocí webového prohlížeče jako obrázek. Tato třída je využívána pro generování grafů ve statistikách.

Třída chat

Řeší zobrazení skrytí chatu, odeslání zprávy, přijmutí zprávy a konzistentní model zpráv. Zprávy přesně zobrazuje podle času odeslání. Pokud čas odeslání je u více zpráv stejný, má přednost zpráva od hráče, který se připojil do hry dříve.

Třída input_bubble

Implementuje zobrazování hlášení v případě nevalidních vstupů. Nad vstupem, v případě pokusu o nevalidní vstup, na pár sekund zobrazí bublinu informující o neplatném vstupu.

Třída storage

Slouží pro ukládání a čtení dat aplikace do prohlížeče. Je využívána pro ukládání posledního přiděleného UUID, přezdívky hráče, profilového obrázku hráče a posledního UUID identifikující posledního zakladatele hry ke kterému byl uživatel připojen.

Třída map

Slouží pro vytváření deterministické mapy na základě vstupního semínka. Vykresluje mapu pomocí softwarového rendereru.

Třída messagebox

Vytváří a zobrazuje uživatelské rozhraní pro zadání jména, profilového obrázku, vytvoření nové hry, odpočet do začátku nové hry a zobrazuje v uživatelském rozhraní výsledky hry.

Třída money

Periodicky vypočítává aktuální množství peněz hráče. Pokud hráč provede nějakou akci i protivník, která vyžaduje odečtení peněz, vypočítá zvlášť stav peněz daného hráče přesně pro okamžik provedení akce, který předá dál ke kontrole, zda daný hráč má nárok provést operaci. Pokud má nedostatek peněz je akce dále zrušena.

Třída peer_network

Zajišťuje propojení všech peerů a odesílání zpráv mezi všemi propojenými peery. Řeší nechtěná propojení z jiných aplikací.

Třída protocol

Obsahuje implementaci odesílání většiny zpráv a čtení přijatých zpráv. Ve třídě je krásně vidět formát jednotlivých zpráv.

Třída statistic

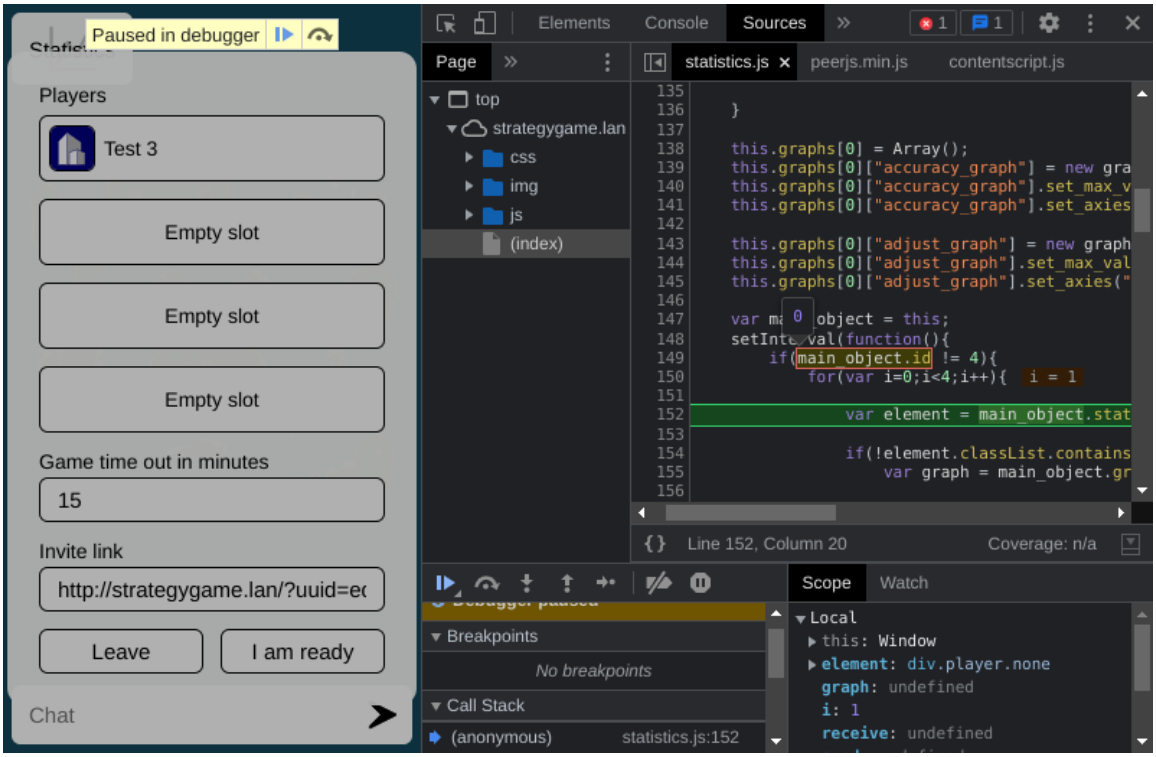
Umožňuje zobrazit statistiky synchronizace času, přenesených dat, latence sítě a stav všech peerů. Pro vytváření grafů statistik využívá třídu 4.10.

Třída game_time

Obsahuje implementaci synchronizace času. Vytváří rozhraní, které umožňuje získat synchronizované herní hodiny. Periodicky provádí synchronizaci času k serveru v intervalu jedné sekundy.

4.11 Debuggování aplikace

Pro debuggování aplikace jsem využil vývojářské nástroje aplikace Chromium. V těchto nástrojích jde velmi efektivně krokovat vykonávaný kód a je velmi dobře vidět jaké data obsahují proměnné. Ukázka z debuggování je vyobrazena na obrázku 4.8.



Obrázek 4.8: Debuggování pomocí vývojářských nástrojů v aplikaci Chromium

4.12 Zprovoznění na webovém serveru s přesným časem

Pro umožnění aplikaci používat přesnější čas, je nutné stáhnout knihovnu PeerJS na stejný server, na kterém je aplikace hostována. Dále server musí poskytovat obsah zabezpečeně přes protokol HTTPS a musí odesílat na klienta hlavičky:

```
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Embedder-Policy: require-corp
```

Pokud se k serveru přistupuje přes localhost, tak server nemusí obsah dodávat zabezpečeně pro přesnější čas. Bez serveru není možné zpřesnit čas v aplikaci.

4.13 Umístění na webový server

Aplikace je zprovozněná na webové drese https://www.stud.fit.vutbr.cz/~xkulis03/strategy-game.

Kapitola 5

Ověřování funkčnosti aplikace

Testování funkčnosti konzistence kolaborativní scény jsem prováděl pomocí více metod.

5.1 Standardní podmínky

První typy testů probíhaly za standardních podmínek. Během těchto testů byly odhaleny některé chyby, které nastávaly jen se specifickým semínkem pro deterministickou simulaci. Například s některými semínky nastávalo během deterministické simulace dělení nulou a aplikace díky tomu zamrzla na všech zařízeních současně. Tuto chybu se povedlo opravit díky debuggovacím nástrojům aplikace Chromium.

5.2 Simulovaná latence sítě

U druhé metody jsem za pomoci vytvoření vysoké latence v jednom směru komunikace, způsobil velký rozdíl mezi synchronizovanými hodinami. Díky tomu na jednom zařízení šly hodiny o několik sekund dopředu a šla jednoduše zkoušet konzistence kolaborativní scény v případě nepravdivých predikovaných stavů scény.

Během tohoto testování byly odhaleny chyby v kódu, které způsobovaly nekonzistenci scény.

5.3 Různá zařízení a prohlížeče

Další testy byly prováděny na různých zařízeních s různými prohlížeči, které byly mezi sebou propojeny. Tyto testy měly odhalit nekonzistence způsobené různými platformami a různými implementacemi webových prohlížečů.

Během těchto testů se odhalilo zamrzání monotonických hodin z aplikačního rozhraní Performance, při uzamknutí obrazovky v Chromu na operačním systému android. Na základě tohoto zjištění přibyla detekce zamrznutí monotónních hodin do implementace synchronizace času. Pokud se detekuje zamrazení hodin, je čas herních hodin maximálně do 1 sekundy + latence sítě opraveny.

Dále se odhalily chyby v GUI aplikace na prohlížeči Firefox. Jednalo se například o zobrazování scroll baru v částech na kterých nešlo scrollovat. Všechny tyto chyby bylo opraveny.

5.4 Ověřování funkčnosti synchronizace času

Funkčnost synchronizace času jsem ověřoval na základě informací zobrazovaných ve statistikách sítě a porovnávání času na více zařízeních. Během vývoje díky těmto testům byla odhalena spousta chyb.

Kapitola 6

Výsledky měření

Kapitola se zabývá výsledky měření výsledné aplikace a vysvětluje co je na jednotlivých výsledcích vidět.

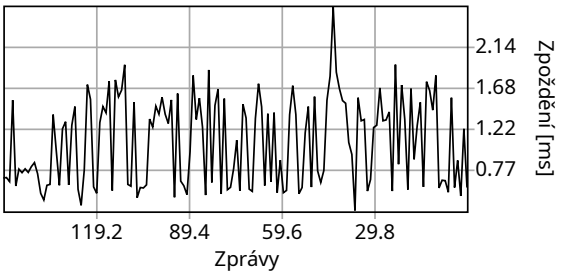
6.1 Synchronizace času a zpoždění mezi zprávami

Zpoždění bylo měřeno k jednotlivým zprávám aplikace (ne k doručení jednoho paketu), aby byla vidět reálná odezva pro aktualizaci kolaborativní scény. Odezva byla měřena na základě synchronizovaných herních hodin. Komunikace probíhala mezi zařízeními šifrovaně. Měření bylo prováděno s použitím webového serveru, který splňoval podmínky, které jsou nutné pro možnost mít z aplikačního rozhraní přístup k přesnějšímu času, aby byly výsledky přesnější. Pro běh aplikace byl použit prohlížeč Chromium, který poskytuje přesnější monotónní čas z aplikačního rozhraní, než Firefox. V některých měřeních bylo v prohlížečích vypnuto skrývání lokální IP adresy za mDNS záznam, aby bylo dosaženo vyšší přímé konektivity mezi prohlížeči v různých podsítích, přes které mDNS záznam neprošel.

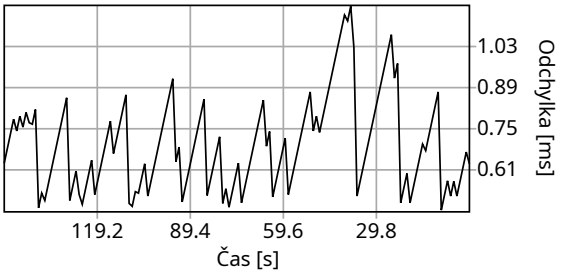
Přesnost synchronizace času zobrazuje nejhorší možný rozdíl k referenčním hodinám, který by nastal, v případě kdyby odezva v jednom směru komunikace byla nulová, což je nereálné. Reálně má synchronizace hodin mnohem větší přesnost. Nepřesnost synchronizace je způsobena rozdílnou dobou odeslání a přijmutí zprávy, více v sekci 2.6.

Jedno zařízení

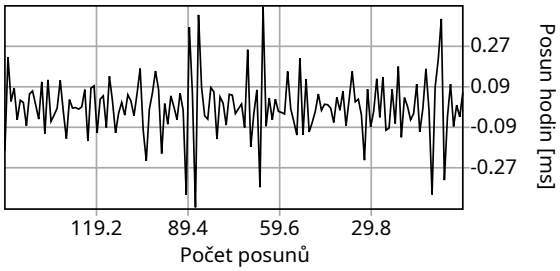
První měření vyobrazené na 6.1, proběhlo na jednom zařízení, na kterém byly otevřeny dvě okna prohlížeče s otevřenou aplikací. Měření bylo provedeno, aby bylo vidět jakých nejlepších výsledků je přibližně aplikace schopná dosáhnout.



(a) Zpoždění



(b) Maximální odchylka hodin



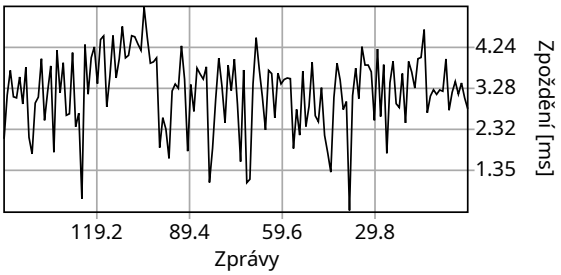
(c) Posouvání hodin

Obrázek 6.1: Synchronizace času a zpoždění na localhostu

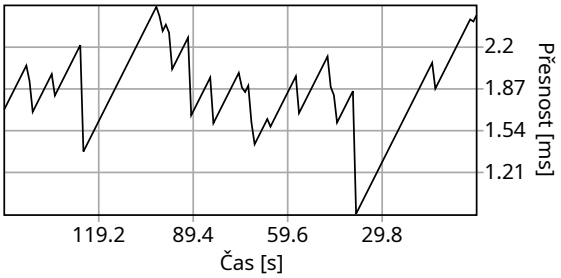
V testu byla průměrná latence 1.12 ms.

Gigabytová síť

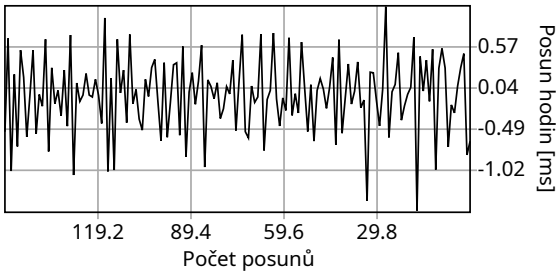
Druhé měření zobrazené na 6.2, probíhalo na gigabytové síti skrze dva mosty, které byly realizovány zařízeními Mikrotik hap ac^2 a Mikrotik hap ac^3. Během měření přes spojení těchto dvou zařízení probíhal i jiný provoz.



(a) Zpoždění



(b) Maximální odchylka hodin



(c) Posouvání hodin

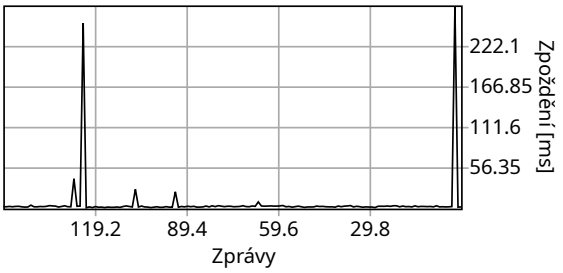
Obrázek 6.2: Synchronizace času a zpoždění na lokální síti

Na výsledcích je vidět jen zvýšení latence mezi zařízeními, není vidět nutnost znovu odesílat ztracené pakety.

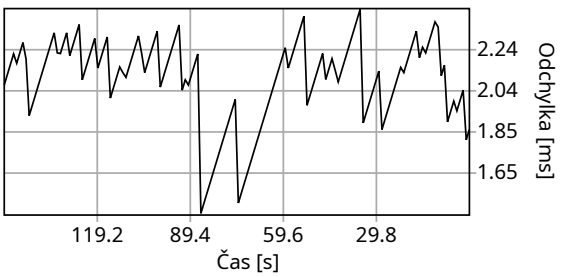
Průměrná latence byla 3,173 milisekund.

5GHz WiFi

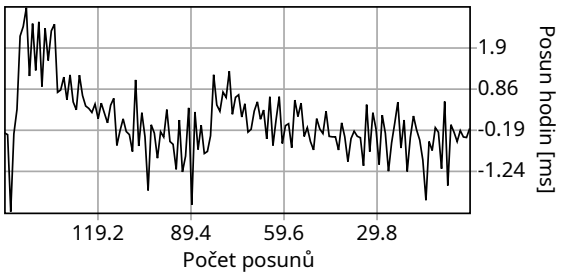
Třetí měření probíhalo na 5GHz WiFi, která vysílala na málo používaných frekvencích, okolními WiFi přístupovými body. Síla signálu kolísala okolo -70 dBm.



(a) Zpoždění



(b) Maximální odchylka hodin



(c) Posouvání hodin

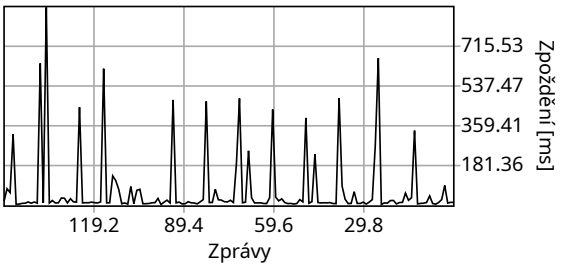
Obrázek 6.3: Synchronizace času a zpoždění na 5GHz WiFi se silou signálu -70dBm

Na výsledcích je vidět občas vysoká latence, která byla způsobena ztrátou paketu a nutností znovu odeslání dat. Občasná ztráta paketů nějak zásadně neovlivnila odchylku synchronizovaných hodin. Každopádně ovlivnila plynulost hry v případě, kdy byla odeslána zpráva o aktualizaci scény a některá část se z ní ztratila a bylo ji nutné znovu odeslat.

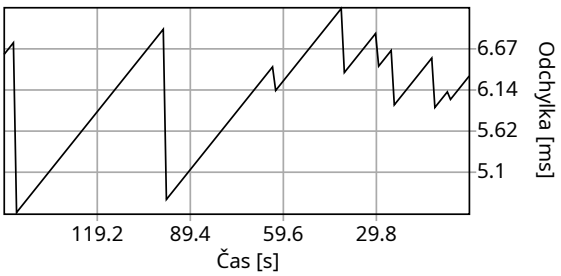
V třetím testu byla průměrná latence 5,545 milisekund.

Dva WiFi routery

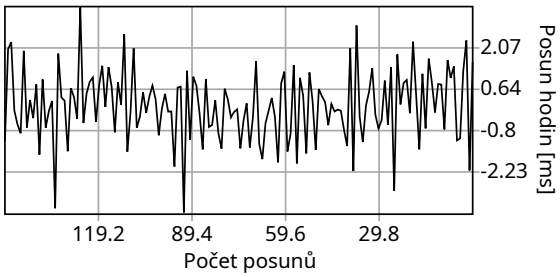
Čtvrté měření probíhalo přes síť tvořenou dvěma WiFi routery. Jako WiFi routery byly použity zařízení Mikrotk hap ac3 a starý ASUS RT-N12LX. Síla signálu mezi WiFi routery kolísala okolo -90dBm. K jednomu WiFi routeru byl připojen počítač přes síťový kabel a k druhému byl připojen přes WiFi se silou signálu, která kolísala okolo -88 dBm.



(a) Zpoždění



(b) Maximální odchylka hodin



(c) Posouvání hodin

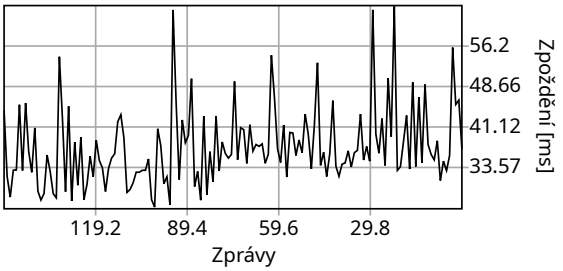
Obrázek 6.4: Synchronizace času a zpoždění přes dvě WiFi zařízení

Z měření je vidět větší nutnost znovu odesílat pakety v některých zprávách. Přesnost synchronizace času se o mnoho nezhoršila.

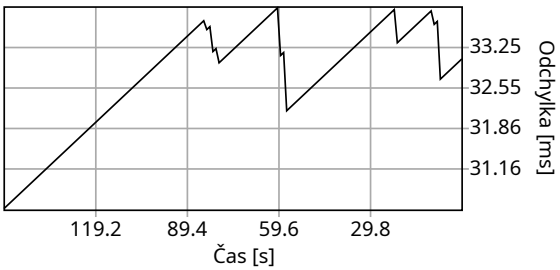
Průměrná latence činila 66,87 milisekund.

WiFi ISP a mobilní data

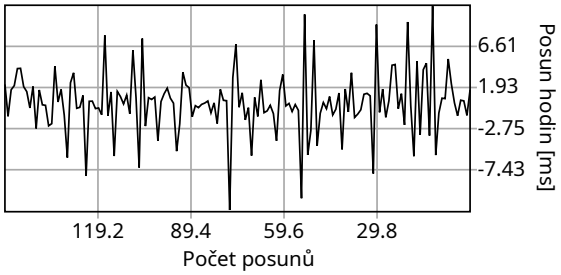
Páté měření probíhalo mezi místním WiFi ISP poskytovatelem a poskytovatelem mobilních dat.



(a) Zpoždění



(b) Maximální odchylka hodin



(c) Posouvání hodin

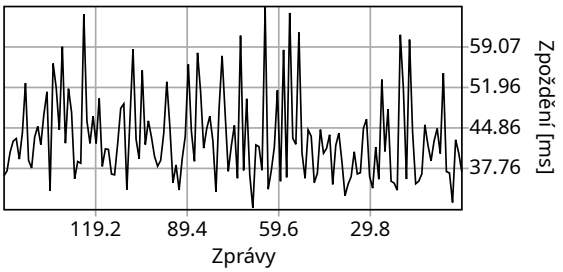
Obrázek 6.5: Synchronizace času a zpoždění přes 2 poskytovatele internetu

Jak je vidět, tak nepřesnost synchronizace hodin se zvětšila oproti předchozím měřením. Nicméně odezva mezi zprávami nemá tak velké odchylky oproti měření se slabým WiFi signálem. Díky tomu je odezva mezi některými zprávami lepší a hra působí plynuleji. Zprávy nejsou tak často zamítány z důvodu konzistence a predikce je přesnější.

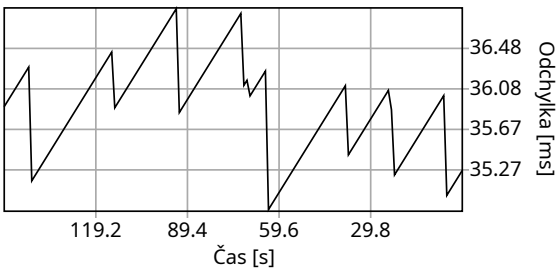
V tomto testu byla průměrná latence 36,69 milisekund.

WiFi ISP a mobilní data s použitím TURN serveru

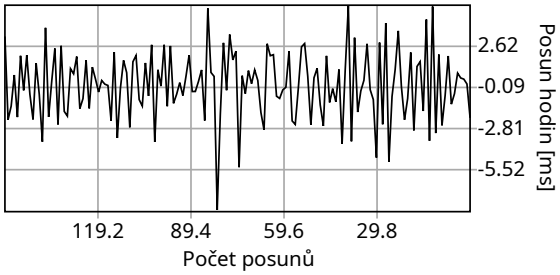
Šesté měření probíhalo na stejné síti jako měření páté s rozdílem, že zařízení mezi sebou komunikovali přes TURN server. TURN server se používá v případě, když selže UDP děrování a zařízení nemohou spolu komunikovat napřímo bez prostředníka, více v sekci 2.3. Použitý TURN server ležel v sousední zemi a měl velmi dobrou internetovou konektivitu.



(a) Zpoždění



(b) Maximální odchylka hodin



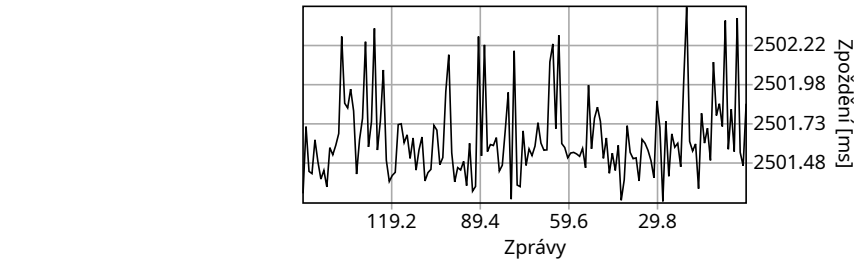
(c) Posouvání hodin

Obrázek 6.6: Synchronizace času a zpoždění přes 2 poskytovatele internetu s vytvořeným propojením přes TURN server

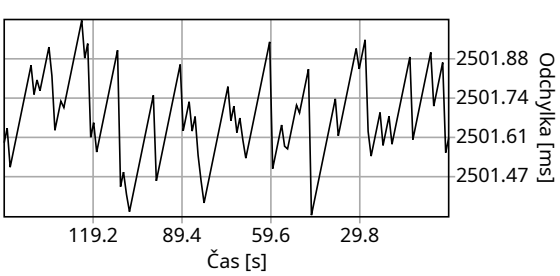
Díky dobré konektivitě TURN serveru se latence zvýšila jen o 6.58 milisekund. Průměrná latence byla 43,27 milisekund. Průběh hry působil stejně jako u předchozího měření.

Umělá latence 2500 milisekund

Sedmé měření probíhalo na lokální síti s uměle nastavenou latencí na 2500 milisekund v obou směrech komunikace. Tady toto měření jsem prováděl, abych ověřil, jak se aplikace vypořádá s vysokou latencí.



(a) Zpoždění



(b) Maximální odchylka hodin

(c) Posouvání hodin

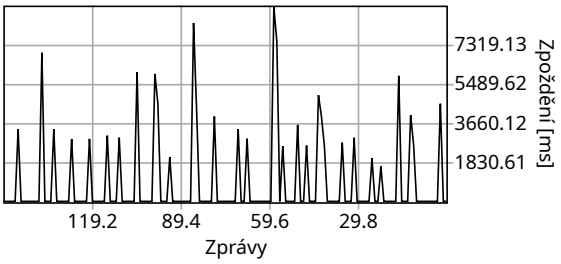
Obrázek 6.7: Synchronizace času a zpoždění s uměle vytvořenou latencí

Hra působila velmi neplynule. Například při hraní nastávaly situace, kdy si hráč koupil firmu a po přibližně 2,5 sekundách mu byla odebrána, jelikož přišla zpráva od druhého hráče s informací, že si firmu koupil dříve. Na grafech je vidět vysoká nepřesnost synchronizace hodin. Nicméně díky tomu, že latence byla v obou směrech stejně vysoká, tak hodiny měly stejnou odchylku, jako na lokální síti bez uměle přidané latence. V grafech je uvedená vysoká odchylka kvůly tomu, že aplikace si nemuže být jistá, zda je vysoká latence jen v jednom směru komunikace. Podronější informace jak synchronizace probíhá je napsáno v podkapitole 2.6 sekce NTP.

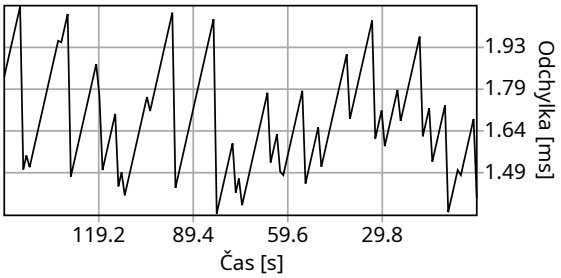
Průměrná latence byla 2501.693 milisekund.

Uměle vytvořená ztrátovost paketů 90%

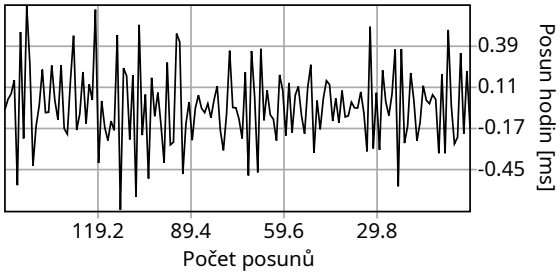
Poslední měření probíhalo na lokální síti s uměle vytvořenou ztrátovostí paketů na 90% v obou směrech. Toto měření jsem prováděl, abych ověřil funkčnost aplikace v prostředí s velmi vysokou ztrátovostí paketů.



(a) Zpoždění



(b) Maximální odchylka hodin



(c) Posouvání hodin

Obrázek 6.8: Synchronizace času a zpoždění s uměle vytvořenou ztrátovostí paketů

Tady toto měření zvýraznilo efekt, který byl vidět na měření s dvěma WiFi zařízeními se špatným signálem. Synchronizace času byla velmi přesná, ale zprávy se musely velmi často posílat znovu a díky tomu byla hra velmi neplynulá a působila velmi podobně jako u předchozího měření s tím rozdílem, že zprávy se opožďovaly různou dobou oproti předchozím konstantním 2,5 sekundám.

6.2 Objem přenesených dat

Na obrázku 6.9 a 6.10 je vidět výpis všech statistických souborů, které je nutné přenést pro funkčnost celé aplikace. Některé vektorové obrázky nejsou nutné pro start aplikace a jsou přenášeny až během práce s aplikací.

Name	Status	Type	Initiator	Size
peerjs.min.js	200	script	(index)	37.3 kB
favicon.ico	200	vnd.microsoft.icon	Other	10.0 kB
peer_network.js	200	script	(index)	4.4 kB
statistics.js	200	script	(index)	3.5 kB
map.js	200	script	(index)	3.1 kB
messagebox.js	200	script	(index)	3.1 kB
companies.js	200	script	(index)	2.9 kB
chat.js	200	script	(index)	2.1 kB
game.js	200	script	(index)	2.1 kB
company_list.js	200	script	(index)	2.0 kB
money.js	200	script	(index)	1.9 kB
protocol.js	200	script	(index)	1.9 kB
control_panel.js	200	script	(index)	1.7 kB
time_sync.js	200	script	(index)	1.7 kB
graph.js	200	script	(index)	1.7 kB
control_panel.css	200	stylesheet	(index)	1.5 kB
main.js	200	script	(index)	1.4 kB
trophy_3.svg	200	svg+xml	messagebox.js:…	1.3 kB
trophy_2.svg	200	svg+xml	messagebox.js:…	1.3 kB
trophy_1.svg	200	svg+xml	messagebox.js:…	1.3 kB
input_bubble.js	200	script	(index)	1.3 kB
messagebox.css	200	stylesheet	(index)	1.3 kB
buy.svg	200	svg+xml	control_panel.css	1.3 kB
chat.svg	200	svg+xml	control_panel.css	1.2 kB
account.svg	200	svg+xml	messagebox.js:81	1.1 kB
chat.css	200	stylesheet	(index)	1.1 kB
my.svg	200	svg+xml	control_panel.css	1.1 kB
enemy.svg	200	svg+xml	control_panel.css	1.1 kB
lvl1.svg	200	svg+xml	company_list.js:75	1.1 kB
lvl2.svg	200	svg+xml	company_list.js:75	1.1 kB
companies.svg	200	svg+xml	control_panel.css	1.1 kB
lvl3.svg	200	svg+xml	company_list.js:75	1.1 kB
statistic.css	200	stylesheet	(index)	1.1 kB
strategygame.lan	200	document	Other	1.0 kB
map.css	200	stylesheet	(index)	923 B
main.css	200	stylesheet	(index)	810 B
input_bubble.css	200	stylesheet	(index)	799 B
local_storage.js	200	script	(index)	772 B
map_enemy.svg	200	svg+xml	map.js:378	715 B
map_my.svg	200	svg+xml	map.js:375	715 B
congruential_generator.js	200	script	(index)	710 B
map.svg	200	svg+xml	map.js:339	704 B
statistics.svg	200	svg+xml	statistic.css	694 B
close.svg	200	svg+xml	statistic.css	645 B
send.svg	200	svg+xml	chat.css	619 B
scroll_bar.css	200	stylesheet	(index)	565 B

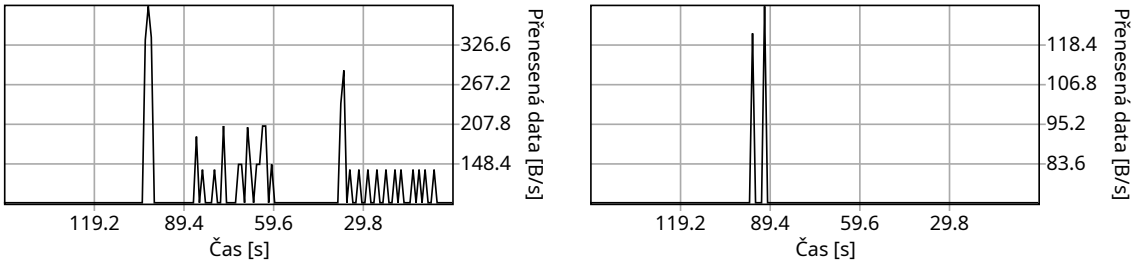
Obrázek 6.9: Přenesená data pro načtení celé aplikace bez cache

Name	Status	Type	Initiator	Size
strategygame.lan	200	document	Other	1.0 kB
peerjs.min.js	200	script	strategygame.la…	(memory cache)
statistics.js	200	script	(index)	(memory cache)
peer_network.js	200	script	(index)	(memory cache)
companies.js	200	script	(index)	(memory cache)
messagebox.js	200	script	(index)	(memory cache)
map.js	200	script	(index)	(memory cache)
favicon.ico	200	vnd.microsoft.icon	Other	(disk cache)
protocol.js	200	script	(index)	(memory cache)
money.js	200	script	(index)	(memory cache)
control_panel.js	200	script	(index)	(memory cache)
chat.js	200	script	(index)	(memory cache)
company_list.js	200	script	(index)	(memory cache)
game.js	200	script	(index)	(memory cache)
time_sync.js	200	script	(index)	(memory cache)
control_panel.css	200	stylesheet	(index)	(disk cache)
graph.js	200	script	(index)	(memory cache)
main.js	200	script	(index)	(memory cache)
messagebox.css	200	stylesheet	(index)	(disk cache)
input_bubble.js	200	script	(index)	(memory cache)
chat.css	200	stylesheet	(index)	(disk cache)
statistic.css	200	stylesheet	(index)	(disk cache)
local_storage.js	200	script	(index)	(memory cache)
map.css	200	stylesheet	(index)	(disk cache)
trophy_3.svg	200	svg+xml	messagebox.js:…	(disk cache)
trophy_2.svg	200	svg+xml	messagebox.js:…	(disk cache)
trophy_1.svg	200	svg+xml	messagebox.js:…	(disk cache)
buy.svg	200	svg+xml	control_panel.css	(disk cache)
chat.svg	200	svg+xml	control_panel.css	(disk cache)
account.svg	200	svg+xml	messagebox.js:81	(memory cache)
lvl1.svg	200	svg+xml	company_list.js:75	(disk cache)
lvl2.svg	200	svg+xml	company_list.js:75	(disk cache)
my.svg	200	svg+xml	control_panel.css	(disk cache)
enemy.svg	200	svg+xml	control_panel.css	(disk cache)
companies.svg	200	svg+xml	control_panel.css	(disk cache)
lvl3.svg	200	svg+xml	company_list.js:75	(disk cache)
input_bubble.css	200	stylesheet	(index)	(disk cache)
main.css	200	stylesheet	(index)	(disk cache)
congruential_generator.js	200	script	(index)	(memory cache)
map_enemy.svg	200	svg+xml	map.js:378	(disk cache)
map_my.svg	200	svg+xml	map.js:375	(disk cache)
map.svg	200	svg+xml	map.js:339	(disk cache)
statistics.svg	200	svg+xml	statistic.css	(memory cache)
close.svg	200	svg+xml	statistic.css	(memory cache)
send.svg	200	svg+xml	chat.css	(memory cache)
scroll_bar.css	200	stylesheet	(index)	(disk cache)

Obrázek 6.10: Přenesená data pro načtení celé aplikace s cache

Pro načtení statických souborů aplikace prohlížečem bylo přeneseno s použitím komprimace gzip a cache pamětí na protokolu HTTP/1.1 1030 bajtů dat. Bez použití cache paměti bylo přeneseno 110975 bajtů (první načtení aplikace prohlížečem). Když se nezapočítá komprimace dat provedená HTTP serverem, bylo by nutné přenést bez cache paměti, popřípadě při prvním otevření aplikace 306605 bajtů dat.

Přes komunikační kanál probíhá neustálá synchronizace času v intervalu 1 sekundy. Tato komunikace vyžaduje z klienta odeslání přibližně 89 bajtů dat na aplikační vrstvě na server a stáhnutí 72 bajtů dat (data se přenáší sterilizovaná ve formátu JSON a mohu měnit s časem svojí velikost, díky reprezentaci v řetězci). Během probíhající hry jsou ještě přenášena data, když nějaký účastník pošle zprávu v chatu, koupí, prodá, aktualizuje firmu.



(a) Odeslaná data

(b) Přijatá data

Obrázek 6.11: Přenesená data na konci hry

Během hry probíhající 15 minut s dvěma účastníky, bylo odesláno na zakladatele hry 86114 bajtů a přijato od něho 66438 bajtů dat. Tvůrce hry měnil stav svých firem méně často. Přes chat nebyla posílána žádná data. Na obrázku 6.11 je vidět množství přenesených dat ke konci hry (posledních 150 sekund).

Kapitola 7

Závěr

Cílem práce bylo seznámit čtenáře se základními metodami kolaborativního sdílení dat a aplikačními rozhraními prohlížečů pro komunikaci přes paketovou síť a na základě těchto informací navrhnout strategickou hru a její konzistentní model kolaborativní scény pro kolaborativní sdílení dat. Navržený model a hru následně implementovat a ověřit správnou funkčnost hry a jejího konzistenčního modelu. Provést měření latence v této hře při různých podmínkách a vyhodnotit její funkčnost při zhoršených podmínkách sítě. Dále změřit množství přenesených dat aplikací.

Vytvořená aplikace používá webové aplikační rozhraní WebRTC, provádí techniky děrování NATu pro co největší snížení latence sítě. Využívá aktivní replikace a provádí na všech zařízeních deterministickou simulaci na základě semínka a synchronizovaných hodin. Díky tomu se na všech zařízeních objevují nové firmy ve hře téměř ve stejný čas, téměř bez známek latence.

Vytvořená aplikace se dokázala vyrovnat s horšími síťovými podmínkami. Při měření se ukázalo, že při velké ztrátě paketu na síti dochází k zhoršení plynulosti hry a nastává častěji rušení predikovaného stavu kolaborativní scény. Při zvýšení latence se zase zhoršuje přesnost synchronizovaných hodin.

Implementovanou aplikaci jde neustále vylepšovat. Například budoucím rozšířením by mohla být implementace zotavení rozehrané hry po pádu aplikace, případně spojení. Dodání dalších herních mechanismů, které by vylepšovaly hratelnost.

Literatura

[1] Bishop, M. *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Internet-Draft draft-ietf-quic-http-34. Internet Engineering Task Force, únor 2021. Work in Progress. Dostupné z: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34.

[2] Cullen Jennings, J.-I. B. A. B. D. C. B. A. N. B. A. T. B. WebRTC 1.0: Real-Time Communication Between Browsers. *All Standards and Drafts - W3C* [online]. ©2021 [cit. 2022-04-16]. Dostupné z: https://www.w3.org/TR/2021/REC-webrtc-20210126/.

[3] Fablet, Y., Uberti, J., Borst, J. D. a Wang, Q. *Using Multicast DNS to protect privacy when exposing ICE candidates*. Internet-Draft draft-ietf-mmusic-mdns-ice-candidates-03. Internet Engineering Task Force, prosinec 2021. Work in Progress. Dostupné z: https://datatracker.ietf.org/doc/html/draft-ietf-mmusic-mdns-ice-candidates-03.

[4] Grigorik, I. *High-performance browser networking*. 3. vyd. O'Reilly Media, Inc., ©2013. ISBN 978-1-449-34476-4. Dostupné z: https://hpbn.co/.

[5] Ilya Grigorik, J. M. High Resolution Time. *All Standards and Drafts - W3C* [online]. ©2022 [cit. 2022-04-26]. Dostupné z: https://www.w3.org/TR/hr-time/.

[6] Kleppmann, M. *Distributed Systems* [online]. 2021-2022 [cit. 2022-04-8]. 91 s. Dostupné z: https://www.cl.cam.ac.uk/teaching/2122/ConcDisSys/dist-sys-notes.pdf.

[7] MDN contributors. XMLHttpRequest. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-03-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest.

[8] MDN contributors. Fetch API. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-03-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

[9] MDN contributors. EventSource. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-03-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/EventSource.

[10] MDN contributors. The WebSocket API (WebSockets). *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-03-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

[11] MDN CONTRIBUTORS. WebRTC API. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-03-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API.

[12] MDN CONTRIBUTORS. Ajax. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-03-31]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX.

[13] MDN CONTRIBUTORS. Writing WebSocket servers. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-04-4]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers.

[14] MDN CONTRIBUTORS. WebRTC connectivity. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-04-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity.

[15] MDN CONTRIBUTORS. Window.localStorage. *MDN Web Docs* [online]. ©1998–2022 [cit. 2022-04-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

[16] PETR PERINGER, M. H. *Modelování a simulace* [online]. 2021 [cit. 2022-04-24]. 332 s. Dostupné z: http://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf.

[17] PEČIVA, J. *Active Transactions in Collaborative Virtual Environments*. Faculty of Information Technology BUT, 2007. 127 s. ISBN 978-80-214-3549-0. Dostupné z: https://www.fit.vut.cz/research/publication/8571.

[18] REITER, A. a MARSALEK, A. WebRTC: Your Privacy is at Risk. In: *Proceedings of the Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, 2017, s. 664–669. SAC '17. DOI: 10.1145/3019612.3019844. ISBN 9781450344869. Dostupné z: https://doi.org/10.1145/3019612.3019844.

[19] SEAH, D., LEONG, W. K., YANG, Q., LEONG, B. a RAZEEN, A. Peer NAT Proxies for Peer-to-Peer Games. In: *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2009. NetGames '09. ISBN 9781424456055.

[20] STEEN, A. S. T. Maarten van. *Distributed Systems*. 3. vyd. Maarten van Steen, ©2017. ISBN 978-90-815406-2-9. Dostupné z: https://www.distributed-systems.net/index.php/books/ds3/ds3-ebook/.

[21] WEI, Y., YAMADA, D., YOSHIDA, S. a GOTO, S. A New Method for Symmetric NAT Traversal in UDP and TCP. Leden 2008, s. 8. Dostupné z: https://www.researchgate.net/publication/228411948_A_New_Method_for_Symmetric_NAT_Traversal_in_UDP_and_TCP.

Příloha A

Obsah paměťového média

- `src` – zdrojové soubory aplikace
- `doc` – zdrojové soubory této bakalářské práce
- `sitova_komunikace_a_kolaborativni_sdileni.pdf` – text této bakalářské práce
- `apache2_src` – zdrojové soubory aplikace s knihovnami a nastaveními pro nasazení na webový server apache2

Příloha B

Přílohy k práci v digitální podobě

- https://gitlab.com/kulisekvojtech/strategy-game – repozitář se zdrojovými kódy
- https://merlin.fit.vutbr.cz/wiki/index.php/Graphics_Projects_2022 – umístění na Wiki@Merlin
- https://www.stud.fit.vutbr.cz/~xkulis03/strategy-game – běžící aplikace na webu

Příloha C

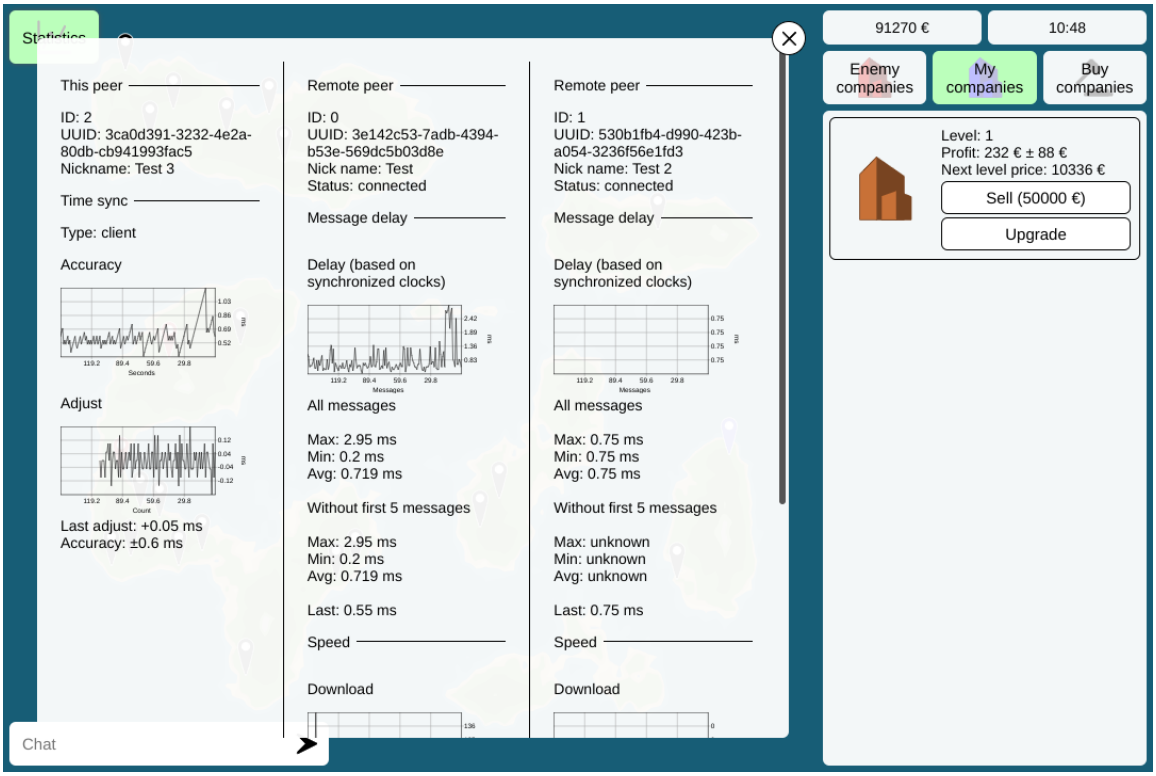
Snímky aplikace

C.1 Verze pro desktopy

Obrázek C.1: Úvodní obrazovka hry

Obrázek C.2: Nová hra

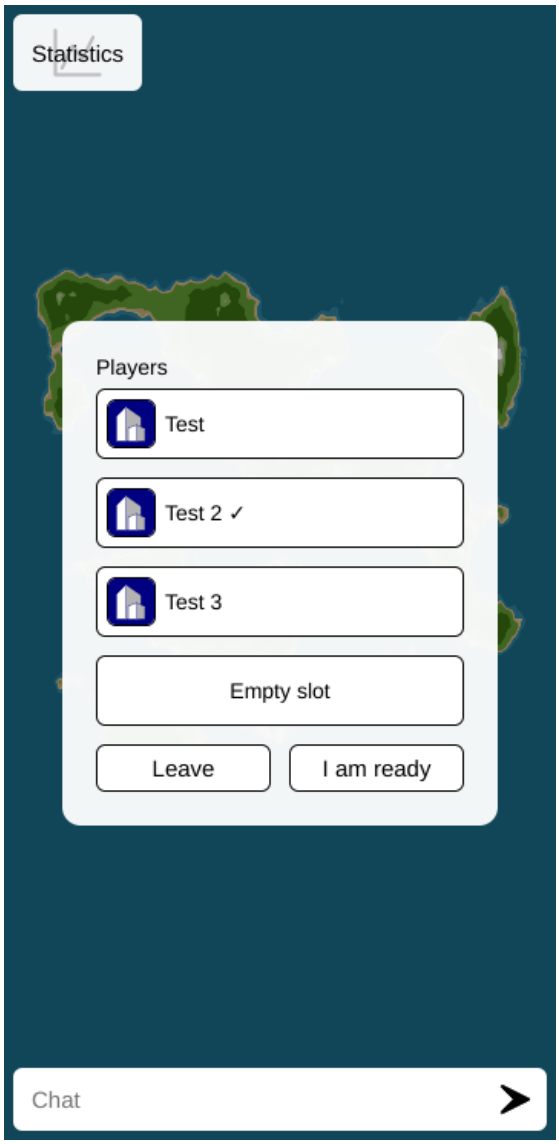
Obrázek C.3: Průběh hry



Obrázek C.4: Statistiky sítě

C.2 Verze pro mobilní telefony

Obrázek C.5: Úvodní obrazovka hry

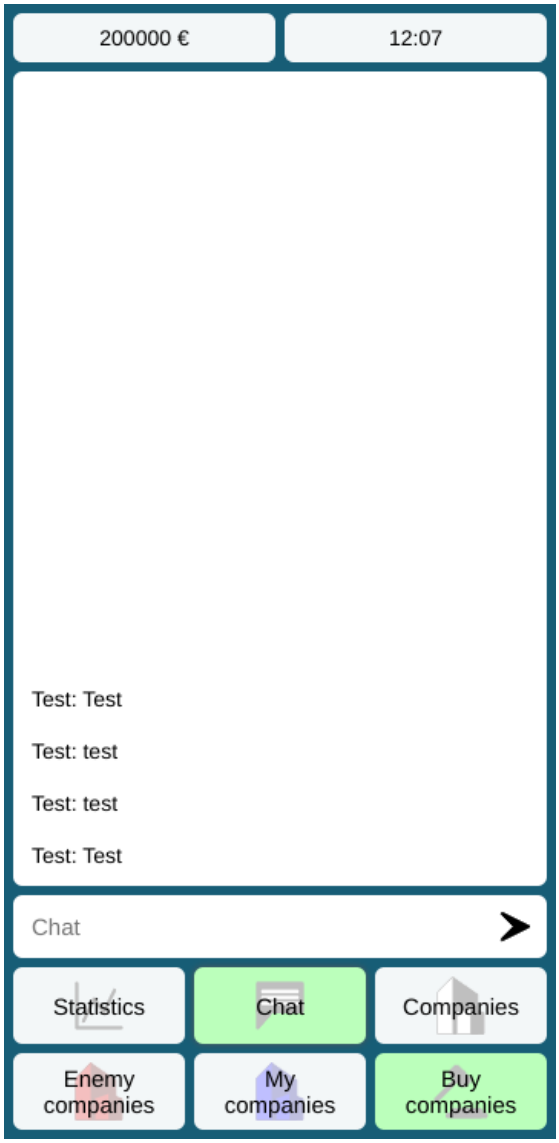


Obrázek C.6: Nová hra

Obrázek C.7: Průběh hry

Obrázek C.8: Zobrazení firem ke koupení

Obrázek C.9: Zobrazení vlastněných firem



Obrázek C.10: Chat