Gymnázium, Praha 6, Arabská 14
předmět programování, vyučující Ing. Daniel Kahoun

Simulátor stolní karetní hry pro více hráčů na síti

ročníkový projekt

Jan Hlavnička, 2.F

2025/26

Abstrakt

Your abstract.

Poděkování

Chtěl bych poděkovat především mému vedoucímu práce ing. Danieli Kahounovi za vedení práce a vstřícnost při konzultacích. Dáloe bych chtěl poděkovat své rodině a kamarádům za to, že mojí hru spoustakrát vyzkoušeli a upozornili na její nedostatky.

Prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne _____

_____

Jan Hlavnička

Obsah

1 Úvod

Tady bude úvod

2 O hře Bang!

Dle wikipedie: Bang! je populární karetní hra na motivy Divokého západu. Hru v roce 2004 vytvořil ital Emiliano Sciarra, vydalo ji nakladatelství DV Giochi. Pro Českou repuliku *Bang!* vydává Albi. V České republice se stal velice oblíbená [33].

2.1 Pravidla hry

Hra implementuje upravenou verzi pravidel. Ta bude v této kapitole popsána. Celé originální pravidla jde najít (odkaz).

2.1.1 Příprava hry

Na začádku hry si každý hráč vybere jednu ze dvou vylosovaných postav. Tato postava mu zůstane po celou dobu hry. Postava má svůj efekt a počet životů. Hráč také dostane roli, kterou (pokud není šerif) ostatní nevidí. Každému hráči bude podle postavy přidán počet karet.

2.1.2 Cíl hry

Cíl hry má každý napsaný na své postavě – šerif s pomocníky musí zabít bandity, banditi šerifa a odpadlík všechny hráče. Hráč umírá tím, že mu dojdou životy a nemá žádné *pivo*.

2.1.3 Průběh kola

Začíná šerif a hráči se střídají v pořadí ve kterém se připojili. Kolo začíná dostáním dvou karet. V průběhu kola může hráč spálit, zahrát a vyložit libovolný počet karet. Jediná. Vyjímkou je karta *bang*, jejíž zahráním tah končí. Tah může také končit zmáčknutím **ukončit tah**.

3 Vyzkoušení hry

Kompletní hra včetně klienta, serveru a všech pluginů zmíněných v této práci je dostupná na webu `bang.honzaa.cz`. Zdrojové soubory jsou umístěním v GitHub repozitáři HonzaHlavnicka-/Bang, kde také jdou přidávat *Issues* pro nalezené problémy.

4 Technologie

Na serveru používá Engine Javu 11, která byla vybraná, aby mohl server běžet i na školním serveru, ktrý novější nepodporoval. Java byla zvolena z důvodu, že je na Gymnáziu Arabská vyučována a protože z programovacích jazyků, které autor zná byla pro projekt nejideálnější. Knihovny na serveru se používá převážně *Java websocket*, která řeší serverovou logiku pro připojení k websocketu a vytvoření websocketového serveru. Dále se používa *org.json* pro složitější práci s JSON objekty.

Na straně klienta se používá *React* s *Vite* a *TypeScriptem*. Nejdříve se používal *vanilla JavaScript*, který byl ovšem nevhodný kvůli těžší rozšiřitelnosti, modularitě s chybějící typové kontrole. Tato původní verze se v repozitáři stále nachází, jelikož v základu funguje. Z frameworků byl zvolen právě *React*, protože ho autor používal v té době na jiný projekt a s jiným neuměl pracovat. Navíc byl *React* pro tento typ projektu vhodný a poskytoval dostatečné funkce i moduly. Z modulů byl použit *react-router-dom* pro přepínání mezi stránkami a *react-dom*. Z externích knihoven se používvá *react-dnd* pro implementaci drag & drop logiky a *react-hot-toast* pro zobrazení upozornění.

5 Cíle práce

Cíle práce byli definovány hned na začádku projektu. Byli zvolený tak, aby obsahovali původní hlavní cí – vyrobit online *Bang!* – ale aby se zaměřil primárně na možnost vytvoření celého herního enginu, ne pouze jednoduché hry. Cíle byli formulovány takto:

Cíle a specifikace práce

- Vytvořit síťovou karetní hru s architekturou klient-server, kde server bude zajišťovat logiku hry a klient bude sloužit k interakci hráče.
- Naprogramovat server v jazyce Java, který bude mimo jiné:
  - spravovat herní logiku,
  - kontrolovat platnost akcí hráčů,
  - zajišťovat průběh hry a aplikaci pravidel,
  - komunikovat s klienty pomocí websocketového protokolu.
- Implementovat webového klienta, který poběží v prohlížeči a umožní hráčům:
  - zobrazit mimo jiné karty v ruce a na stole,
  - sledovat stav a informace o ostatních hráčích,
  - provádět akce a ovládat hru prostřednictvím uživatelského rozhraní,
  - a další potřebné informace pro průběh hry.
- Podporovat variaci karetní hry Bang! s možností rozšíření o další hry.
- Otestovat aplikaci s reálnými hráči a vyhodnotit její funkčnost i uživatelskou přívětivost.
- Vypracovat dokumentaci a návod k použití, včetně popisu architektury a zdrojového kódu.

6 Předem stanovená struktura práce

Již na začátku školního roku 2025–26 bylo potřeba stanovit si osnovu a průběhu práce. Byli zvoleny takto:

Průběh zpracování práce

1. vytvoření jednoduchého serveru
2. vytvoření jednoduchého klienta
3. přidání logiky a ovládacích prvků
4. testování hry s dalšími lidmi
5. opravení nedostatků na základě získaných podmětů
6. sepsání dokumentace a průběhu práce
7. přidání dodatečných méně potřebných funkcí
8. publikace práce

Osnova práce

- o hře Bang! (popřípadě jiných implementovaných hrách)
- návod k programu
- průběh tvorby
- dokumentace
- závěr

7 Práce se zdroji

Ve stejné době bylo třeba deklarovat jaké zdroje budou pro projekt použity. Deklarace zněla takto:

> K práci budu používat oficiální pravidla hry, dokumentaci Javy (docs.oracle.com), dokumentaci Reactu (react.dev) a stránky w3schools.com. Dále pro řešení problémů využiji internetových webových stránek jako jsou například stackoverflow, nástrojů umělé inteligence (chatGPT, copilot) a rad od lidí, kteří mají s problémem zkušenost včetně vedoucího práce pana Ing. Daniela Kahouna.

8 Návod k použití

Chcete-li si zahrát hru, navštivte webovou stránku `bang.honzaa.cz`. Vyberte kartu **Vytvořit hru**, vyberte jakou hru chcete hrát, zadejte své jméno a klikněte na tlačítko **Vytvořit**. Poté se připojte k vytvořené hře a zobrazí se vám kód hry. Teto kód pošlete svým přátelům, aby se mohli připojit. Tis se připojí zadáním kódu v kartě **Připojit se ke hře**. Jakmile se připojí všichni hráči, klikněte na tlačítko **Spustit hru** a můžete začít hrát.

Na hracím poli vidíte dole své karty, nad kterými mohou být vyložené karty. Vlevo dole uvidíte svoji postavu roli, své jméno a podobně. Vlevo nahoře uvidíte informace o ostatních hráčích,

například jejich jména, postavy, role a podobně. Uprostřed uvidíte karty na stole a další herní prvky. Jmenovka hráče, která je zrovna žlutá signalizuje, že je tento hráč na tahu.

Pokud jste na tahu, tak můžete podle pravidel hry co hrajete hrát. Chcete-li si líznout kartu, klikněte na balíček karet. Chcete-li zahrát kartu, tak jí pomocí myši nebo prstu přetáhněte na odhazovací balíček. Pokud chcete kartu spálit (tzn. odhodit bez efektu), tak ji přetáhněte na oheň vedle odhazovacího balíčku. Chcete-li kartu vyložit, tak ji přetáhněte na plochu nad svými kartami. Pokud chcete kartu vyložit před jiného hráče, tak ji stačí přetáhnout na něj. Pokud chcete ukončit tah, klikněte na tlačítko **Ukončit tah**.

Pokusíte-li se udělat něco, co nemůžete, například zahrát kartu mimo svůj tah, tak se vám vpravo nahoře ukáže upozornění, které vám řekne, co se stalo.

9 Implementace serveru

Server se dělí do několika balíčků a projektů. Hlavní balíček je `bang`, který propojuje `sdk` se `server`em. Sdk je klíčová součást celého projektu, protože umožňuje pluginům nebýt závislých na konkrétní implementaci serveru a pracovat jen s interfaci. Balíček `server` je poté konkrétní implementací servových tříd.

9.1 `KomunikatorHry` a `SocketServer`

Pro komunikaci s klienty se používá websocket. Pro jeho správu v javě se používá knihovna *Java websocket*. Díky této knihovny stačí dědit třídu WebsocketServer a Implementovat metody jejích akcí. To dělá právě třída `SocketServer`. Třída obsluhuje zachycení výjimek, vytvoření her, načítání pluginů a komunikaci s hráči na mimoherní úrovni. Tou je myšleno stahování informací o instalovaných pluginech, získávání informací o serveru a podobně.

Nejduležitější metodou je `onMessage`, která zpracovává příslušné zprávy. Dokumentace celého protokolu a všech zpráv je dostupná v dokumentech na GitHubu. Tato metoda umí vytvářet hry a spojovat zprávy s příslušními komunikátoryHer, které si drží v mapě `komunikatoryHracu`.

`KomunikatorHry` je třída, která spojuje `SocketServer` a samotnou hru. Drží si seznamy hráčů, tokeny hráčů pro připojení, mapy hráčů se sockety a podobně. Je používaná k tomu, aby mohli servové nebo pluginové metody kominikovat s klientem. Nemusí k tomu znát přímo `Websocket` objekt, protože ten si spojí v metodě `posli()` sám komunikátor. Komunikátor také zpracovává události posílané klientem přes `SocketServer`. Tyto události rovnou předává do metod hráče, kterého se týkají.

9.2 interface pro balíčky `HerniPlugin` a `HerniPravidla`

Engine je navrhnutý tak, že v základu neobsahuje žádnou hru, ale pouze obecné ulity a funkce. Z toho důvodu je potřeba, aby hra načítala jednotlivé pluginy. To zajišťují třídy `NacitacPliuginu` a `SpravceHernichPravidel`. Aby tyto třídy fungovaly, tak musí každý plugin implementovat `HerniPlugin`. Ten vrací název hry, popis hry, odkaz na pravidla a hlavně metoda, která vytvoří `HerniPravidla`. `HerniPravidla` jsou hlavní třída pluginu, jejíchž metody hra volá, aby se chovala jak autor pluginu očekává. Mezi některé její metody, které může autor používat jsou:

9.2.1 `void pripravBalicek(Balicek<Karta> balicek)`

Tato metoda se volá při přípravě hry a měla by naplnit dobírací balíček `Karta`mi.

9.2.2 `public UIPrvek[] getViditelnePrvky()`

`GetviditelneMetody` je metoda, která vrací seznam prvků, které hráči mohou v klientovy vidět. Přestože by měl být server oddělený od logiky klienta, tak je tato metoda potřebná, aby například ve hře, která nepoužívá princip životů nezabíraly na obrazovce místo ukazatele životů.

9.2.3 `boolean hracChceUkoncitTah(Hrac kdo)`

Metoda, která se volá poté, co hráč klikne na tlačítko ukončit tah. Vrací boolean, který určuje, zda hráč tah ukončit může nebo ne. Spousta her hráči neumožní ukončovat tah když chce, protože hráč musí například odhodit kartu, proto by tyto hry měli v metodě `return false`. Jiné hry, ale nechají hráče udělat kolik akcí chce a proto by jen například kontrolovaly jestli nemá hráč v ruce kartu, kterou před koncem tahu musí odhodit.

9.3 `zacalTah()`

`zacalTah()` je metoda, ve které si může plugin připravit vše, co se odehrává na začátku každého tahu.

Kromě těchto metod třída další, které jsou ale pricipiálně podobné a nemá cenu je zde vysvětlovat. Kompletní seznam možností jde najít v javadoc dokumentaci.

9.4 Karta

Karta je abstraktní třída, jejíchž objekt reprezentuje jednu kontrétní kartu ve hře. Každá karta musí mít jméno a identifikátor obrázku (v sočastnosti cesta k obrázku bez přípony souboru). Každému objektu karty je automaticky přiřazeno unikátní id, které umožňuje její identifikaci v rámci serveru.

Karta sama o sobě nejde zahrát ani vyložit. K tomu poslouží interfaci `HratelnaKarta` a `VylozitelnaKarta`.

9.4.1 `HratelnaKarta`

`HratelnaKarta` je interface, který musí implementovat všechny karty, které může hráč zahrát. Obsahuje metodu `boolean odehrat(Hrac kdo)`, která se volá, když se hráč pokusí kartu zahrát. Vrací boolean zda se karta může zahrát (a rovnou již proběhl její efekt), nebo se v současném kontextu zahrát nemůže.

9.4.2 `VylozitelnaKarta`

`VylozitelnaKarta` je interface, který musí implementovat všechny karty, které může hráč vyložit. Obsahuje metodu `boolean vylozit(Hrac kdo,Hrac kym)`, která se volá, když se hráč pokusí kartu vyložit. Vrací boolean zda se karta může vyložit (a rovnou se k tomu připravý všechny náležitosti), nebo se v současném kontextu vyložit nemůže. Pokud vrátí true, tak je na VylozitelnouKartu zavolána ještě její metoda `getEfekt()`, která vrací objekt implementující interface `Efekt`. Ve většině případů se jedná o objekt karty, ale autor pluginu se může rozhodnout, že logiku karty a efektu bude sržet odděleně, například pokud stejné efekty používá i nějaká postava.

9.5

9.6 `NacitacPluginu` a `SpravceHernichPravidel`

`SpravceHernichPravidel` je třída, která si udržuje seznam všech objektů `HerniPlugin`, které jsou na serveru dostupné. Obsahuje metody `getJSONvytvoritelneHry()`, která vrací JSON objekt obsahující informace o všech dostupných pluginech, které se vyvolají z Herních pluginů a jejich metod. Dále obsahuje metodu `vytvorHerniPravidla(int id, Hra hra)`, která podle id z JSONu zavolá vytvoření pravidel na hře. Tato metoda se využívá vždy když uživatel vytvoří novou hru, aby do hry napojil správný plugin a jeho pravidla. Třída dále obsahuje metodu `pregenerj()`, která načte pluginy ze složky `/pluginy/` a uloží je do seznamu. Metaoda pregeneruj se používá také v *static* bloku, aby se pluginy načetly při startu serveru.

Opravdové načítání pluginů provádí třída `NacitacPluginu`. Ta v metodě `nactiPluginy(Path cestaKeSlozce)` otevře požadovanou složku a pomocí `DirectoryStream` najde všechny soubory s příponou `.jar`. Pro každý takový soubor zavolá metodu `nactiPluginyZJARu(Path cestaKJARu)`, která pomocí `URLClassLoader` načte třídy z JARu a pomocí `Enumeration` hledá všechny třídy (soubory s koncovkou `.class`). Tyto třídy si načte `ClassLoader`em a ověří, zda jsou přiřaditelné k `HerniPlugin`, nejsou abstraktní a nejsou rozhraní. Pokud třída splňuje tyto podmínky, tak se vytvoří její instance a přidá do seznamu pluginů. Pokusy o načítání tříd jsou obalené v try bloku, jelikož je v nich mnoho potenciálních chyb, které je potřeba zachytit, aby se načítání pluginů nezastavilo kvůli jednomu špatnému pluginu, například když jeho konstruktor má nějaký parametr, či když nedpovídá stejné versi *Javy* jako server.

```
public static List<HerniPlugin> nactiPluginyZJARu(Path cesta) throws Exception{
    List<HerniPlugin> pluginy = new ArrayList<>();

    URLClassLoader classLoader = new URLClassLoader(
            new URL[]{cesta.toUri().toURL()},
            NacitacPluginu.class.getClassLoader()
    );

    try (JarFile jar = new JarFile(cesta.toFile())) {
        Enumeration<JarEntry> entries = jar.entries();

        while (entries.hasMoreElements()) {
            JarEntry polozkaJARu = entries.nextElement();

            if (polozkaJARu.isDirectory()) {continue;}
            if (!polozkaJARu.getName().endsWith(".class")) {continue;}

            String nazevTridy = polozkaJARu.getName()
                    .replace("/", ".")
                    .replace(".class", "");

            Class<?> c = classLoader.loadClass(nazevTridy);

            if (HerniPlugin.class.isAssignableFrom(c)
                    && !c.isInterface()
                    && !Modifier.isAbstract(c.getModifiers())) {

                HerniPlugin plugin = (HerniPlugin) c.getDeclaredConstructor().
    newInstance();
                pluginy.add(plugin);
            }
        }
    }
    return pluginy;
}
```

Úryvek 1: Ukázka načítání pluginů z JARu (zjednodušeno)

9.7 `Hra` a `HraImp`

`HraImp` je třída implementující interface `Hra`, která obsahuje všechnu obecnou logiku hry, stav hry a přístup k nim pro pluginy. Hru vytváří pomocí tovární metody `vytvor` komunikátor hry,

který ji id pluginu a sebe, aby mohla komunikovat s klientem. Hra si drží seznam hráčů, dobírací a odhazovací balíček, instanci herních pravidel, správce tahu a podobně.

Hra neřeší ani logiku konkrétní hry, karet, balíčku a akcí hráčů, které řeší Herní pravidla nebo specifické třídy. Hra zařizuje zapnutí hry (`setZahajena(true)`), výhru a konec hry (`skoncil Hrac kdo`, `vyhral(Hrac kdo)`), přidávání hráčů a gettery pro ostatní obslužné třídy.

//todo: dopsat

9.8 Správce tahu

`SpravceTahu` je třída, která se stará o správu tahů ve hře. Přestože se to zdá jako jednoduchý úkol o posouvání ukazatele, tak je třída poměrně složitá protože musí podporovat změnu směru, vynechání tahu, dočasné či trvalé přidání hráče, násobič tahu a podobně.

Pro svoji vnitřní logiku používá třídu `Tah`, která reprezentuje jeden tah ve hře. Tah se znovu používá, to znamená, že jeden objekt nereprezentuje konkrétní tah, ale všechny tahy jednoho hráče. Tah pouze obsahuje veřejné promněné, které ukládají hráče, kterého je to tah, zda je tah jednorázový (tzn. po jeho odehrání se vyřadí z fronty, na rozdíl od normálního tahu, který se opakuje každé kolo) a zda je tah dočasně zrušený (Mohlo by se pozžívat například když má hráč za trest ddvě kola nehrát, ale není žádoucí, aby se vyřadil z pořadí.)

Třída `SpravceTahuImp` obsahuje frontu (`ArrayDeque`) s objekty tahu.

Pro změnu směru se používá metoda `zmenaSmeru()`, která neotáčí celou frontu, ale pouze změní příznak směru, podle kterého se pak rozhoduje, zda se z fronty bere zespoda, nebo zezhora.

Protože je pro některé funkce žádoucí mít k dispozici seznam všech hráčů, kteří nejsou vyřazeni ve správném pořadí, tak existuje metoda `getHrajiciHraci()`. Původně měla fungovat tak, že by správce tahu držel jak frontu tahů, tak informace o hráčích, to by ale bylo velmi náchylné k chybám, protože by bylo potřeba synchronizovat hned dvě struktury bez chyby. Proto byla zvolena varianta generování seznamu z fronty tahů. Jelikož je tato metoda poměrně náročná, tak se její výsledek ukládá do proměnné sloužící jako cache a aktualizuje se pouze při změně fronty tahů, tudíž jejím zneaktuálněním. Díky tomu pluginy mohou metodu volat klidně vícekrát za sebou bez většího výkonostního problému.

```
52      @Override
53      public List<Hrac> getHrajiciHraci() {
54          if (!poradiAktualni) {
55              hrajiciHraciCache = new ArrayList<>();
56              for (Tah t : frontaTahu) {
57                  if (!t.docasneZruseny && !t.jednorazovy) {
58                      hrajiciHraciCache.add((HracImp) t.hrac);
59                  }
60              }
61              if (zmenenSmer) {
62                  Collections.reverse(hrajiciHraciCache);
63              }
64              poradiAktualni = true;
65          }
66          return List.copyOf(hrajiciHraciCache);
67      }
```

Úryvek 2: Ukázka získání hrajících hráčů ve správci tahu

Dalším častým problémem je vyřazení hráče a jeho následné vrácení. To zajišťují metody `vyraditHrace(Hrac koho)` a `vratitHrace(Hrac koho`, které v řadě tahů najdo ten správný a přenastaví mu příznak dočasně zrušený. Je třeba rozlišovat vrácení hráče a jeho přidání, jelikož přidání hráče musí vyrobyt nový `Tah` a vybrat mu nové místo ve frontě. Přidání hráče zajišťuje metoda pridatHrace(Hrac koho)

```
194     public void vratitHrace(Hrac koho){
195         for (Tah tah : frontaTahu) {
196             if (tah.hrac.equals(koho)) {
```

```
197                 tah.docasneZruseny = false;
198             }
199         }
200         poradiAktualni = false;
201     }
202     public void pridatHrace(Hrac koho) {
203         frontaTahu.addLast(new Tah(koho, false));
204         poradiAktualni = false;
205     }
```

Úryvek 3: Ukázka vracení hráče a přidání hráče ve správci tahu

Samotná metoda `dalsiHrac()` musí zkontrolovat všechny příznaky a podle nich rozhodnout, který hráč je na tahu. Metoda taky řeší všechny problémové stavy, jako když nemá kdo hrát.

9.9 `Hrac` a `HracImp`

`HracImp` je třída implementující interface `Hrac`, která reprezentuje jednoho hráče ve hře. Obsahuje informace o hráči, jako je jeho jméno, role, postava, počet životů, karty v ruce, karty na stole a podobně. Dále obsahuje metody pro manipulaci s těmito daty.

9.9.1 `boolean odeberZivot()` a `boolean pridejZivot()`

Odeber život je motoda, kterou by měl plugin volat, když hráč přijde o život. Metoda zkontroluje, zda ještě před odebráním života nemá zavolat nějaký efekt a potom zkontroluje, jestli se hráč nezabije. Pokud se zabije, tak se nepůjde do mínusu v počtu životů, ale vrátí se false a zavolá se `HerniPravidla.dosliZivoty(Hrac kdo)`, aby plugin mohl zareagovat a například zařídit smrt hráče. Pokud hráč ještě nezemřel, tak se odebere život, zavolají efekty a metoda vrátí true.

Přidej život je podobná metoda s opačným účinkem. Míso smrti kontroluje, zda se nepřesáhne daný maximum životů, ktré hráč může mít.

9.9.2 `odehranaKarta(String id)`, `vyloitKartu(String id)` a `spalitKartu(String id, String idHrace)`

Tuto metodu volá komunikátor hry, když se hráč rozhodne zahrát kartu. Není určená pro plugin, protože obsahuje všechny kontroly pro vstup od uživatele, navíjako parametr bere *String* id karty a ne samotný objekt, což je v rámci serveru neefektivní.

Metoda si převede id do *int*u, ověří, že je hráč na tahu a pokusí se kartu najít mezi kartami v ruce porovnáváním id. Poté zkontroluje, zda je karta hratelná (instance `HratelnaKarta`), přetypuje ji, zeptá se pravidel jestli ji může zahrát a pokud ano, tak zavolá její metodu `odehrat(Hrac kdo)`. Pokud se karta zahrála, tak ji přesune na odhazovací balíček, provede efekty a upozorní komunikátor hry, který informuje klienty o změně stavu hry. V tomto procesu nastalo spoustu chybových stavů, které vždy předají konkrétní chybu komunikátoru, který ji zobrazí klientovi.

Podobným způsobem funguje také spálení a vyložení karty, které zajišťují metody `vyloitKartu(String id)` a `spalitKartu(String id, String idHrace)`.

10 Implementace pluginů

K enginu bylo vyrobeno několik pluginů – Bang, prší a UNO. Pluginy jsou v repozitáři ve složce `/pluginy/`.

10.1 Bang!

Tento plugin je pro projekt klíčový, jelikož je engine přízpůsobený hlavně jemu. V původní fázi byl dokonce do enginu hardcodován. Teď už je většina jeho prvků přesunuta do jeho herních pravidel.

V Bangu bylo potřeba zařídit, aby měl správně nakonfigurované pravidla. Dále bylo potřeba Implementovat všechny karty, efekty a postavy.

10.2 Prší

Prší je hra, ktera byla kompletně přidána jako první a používala se v pro testování.

Implementace karet byla jednoduchá. Existuje základní karta `PrsiKarta`, která má svoji barvu () a hodnotu (). Karta je Implementace interface `HratelnaKarta`. Metoda volání při odebrání neprovádí žádný speciální efekt, pouze kontroluje předchozí kartu v odhalování balíčku, zda se shoduje hodnotou, barvou, či zda vůbec není instancí Prší karty (To by nastat nemělo, ale kdyby nějaký balíček kombinovat karty, tak se hodí mít definované chování.

Tuto základní kartu implementuje PrsiSedmicka. Ta se liší v tom, že následující hráč si musí lízat, či sedmičku přebít. Naprogramovat tuto kartu samostatně nejde, protože mění stav hry. Z toho důvodu bylo třeba do herních pravidel přidat oro sedmičku metodu. (doplnit)

Dalším dítětem Prší karty je `PrsiSvrsek`. Ten se liší tomu, že jde vložit nad libovolnou kartu, tudíž jeho metoda `odehrat()` vrátíte vždy true. Než to ale udělá, tak u hráče co ji zahrál vyvolá dialog na výběr barvy. Do té doby, než vybere barvu, tak jeho getter pro ostatní karty vrací `null`. Poté co hráč vybere barvu, tak se svršek změní tu svoji "falešnou"barvu na barvu, kterou hráč vybral. Také se ostatním pošle upozornění, jaká barva byla vybraná.

Poslední speciální kartou je `PrsiEso`. Jelikož autor hry naprogramoval prší podle pravid3l, které používá on, tak se eso nepřebíjí. Z toho důvodu karta po zkontrolování podmínek zahrání pouze zavolá `hra.getSpravceTahu().eso()`, jelikož eso je funkce v enginu zabudovaná.

10.3 Uno

Poslední implementovaná hra je UNO, které je pouze starší verzí prší, bez další složité naprogramováné mechaniky. Nejrozlišnější karta je změna směru, ale její funkčnost je zabudovaná ve správci tahu.

Na tomto pluginu je nejzajímavvější, generování obrázků jeho karet pomocí pythonu.

11 Protokol

Pro komunikaci mezi klientem a serverem se používá websocketový protokol. Již od začátku se posílají stringy ve formátech:

```
1 <typZpravy>:<data>
2 <typZpravy>
3 <typZpravy>:<údaj1>,<údaj2>,...,<údajn>
4 <typZpravy>:<json>
```

Úryvek 4: Formáty komunikačních zpráv systému

Tento formát sice nepatří mezi nejefektivnější a řešením jednotlivých bitů by šel velmi zkrátit, ale pro množství poslaných zpráv a jejich velikost by to bylo neznatelné. Takhle je formát jedno-

duchý a přehledný, takže i při testování jde psát zprávy ručně a není potřeba žádný nástroj pro serializaci a deserializaci.

Hlavní problém protokolu je, že nepojmenovává jednotlivé typy zpráv vždy stejným způsobem, což může být matoucí. Tento problém vznikl tím, že se protokol vyvíjel postupně a nebyl od začátku přesně definovaný.

Průběh komunikace začíná první fází tak, že server uvítá klienta, potom klient zažádá o seznam dostupných her, server mu ho dodá. Pak si hráč vybre nějakou akci připojení (vrácení se, vytvoření hry, připojení se ke hře) a řekne své jméno.

```
-> welcome
<- infoHer
-> infoHer:{"verze":"0.0.7","hry":[{"id":0,"jmeno":"UNO!","popis":"Slavná základní
    karetní hra.","url":""}, {"id":1,"jmeno":"šPrí","popis":"Česká čtradiní hra, ve které
     se žsnaíte zbavit švech karet, které žžmete zahrát podle stejné barvy nebo hodnoty."
    ,"url":""}, {"id":2,"jmeno":"Bang!","popis":"Populární hra s cílem zabít ostatní
    čhráe.","url":"https://albi.cz/data/files/products/24327/1603711687-bang-pravidla-
    zakladni-hry.pdf"}]}
<- pripojeniKeHre:123456
<- noveJmeno:Honza
-> pripojenKeHre
```

Úryvek 5: Příklad průběhu 1.fáze komunikace

V další, druhé, fázi už je hráč připojen ke hře, která ale není zahájena. V této fázi se načítají různé informace o hře a hráčích, kteří se mohou i postupně připojovat. Klientovy bude přiřazen znovupřipojovací token, který mu umožní se znovu připojit ke hře, pokud se odpojí a jeho id hráče, aby věděl jaké informace se týkají přímo jeho.

```
-> token:123456_08gs7oLEqthBeaGkelKVAnl-f_CN5YoctfE7rd0nek
-> vyberPostavu:[{"jmeno":"TESTOVACÍ","obrazek":"TESTOVACI","popis":"pouze pro test","
    zivoty":"4"},{"jmeno":"TESTOVACÍ","obrazek":"TESTOVACI","popis":"pouze pro test","
    zivoty":"4"}]
-> novyHrac:{"id":4, "jmeno":"nepojmenovaný čhrá","zivoty":0,"pocetKaret":0,"
    maximumZivotu":0}
-> noveIdHrace:4
-> povoleneUI:["ODHAZOVACI_BALICEK","DOBIRACI_BALICEK"]
-> noveJmeno:4,honza
-> novyHrac:{"id":5, "jmeno":"nepojmenovaný čhrá","zivoty":0,"pocetKaret":0,"
    maximumZivotu":0}
-> noveJmeno:5,Adam
<- setPostava:1
```

Úryvek 6: Příklad průběhu 2.fáze komunikace

Třetí fáze už je samotná hra. Klient posílá akce hráče, zatímco server mu vrací svoje vypočítané reakce, aktualizace stavu hry a akce jiných hráčů.

```
<- zahajeniHry
-> novaKarta:{"jmeno":"red9","obrazek":"uno/red9","id":109}
-> novaKarta:{"jmeno":"yellow6","obrazek":"uno/yellow6","id":136}
-> role:BANDITA
-> novyPocetKaret:5,8
-> hraZacala
-> zahZacal:5
[druhý hrác odehraje]
-> odehrat:5|{"jmeno":"yellow9","obrazek":"uno/yellow9","id":139}
-> novyPocetKaret:5,7
-> tvujTahZacal
[tento hrác si lízne]
<- linuti
-> novaKarta:{"jmeno":"eso","obrazek":"uno/eso","id":148}
-> tahZacal:5
```

Úryvek 7: Příklad průběhu 3. fáze komunikace

12 Implementace klienta

12.1 Struktura

12.2 O aplikaci

Jak již bylo zmíněno, tak klient je napsán pro framework *React* pro web. Obsahuje hlavní soubor `main.tsx`, který obsahuje různé providery a celoaplikacové komponenty. (zde bude odkaz na moduly) Hlavní komponenotou celé aplikace je `App`, která přepíná mezi stránky a řeší lazyloading. Jednotlivé stránky se nacházejí v `/src/pages/`.

Hlavní kontext, který obsahuje jak funkce pro celou aplikaci a všechny data je `GameKontext`. Ten obsahuje jednotlivé funkce pro posílání dat na sever, úpravu dat, akce hráče a podobně. Jeho provider se jmenuje `GameProvider`. V něm nejsou implementovány tyto funkce, ty jsou spolu s logikou příchozích zpráv přemístěny do `gameActions`. Provider ovšem obsahuje efekt, pomocí kterého se připojí k websocket serveru.

12.3 Připojovací stránkka `LoginPage`

Tato stránka by pro uživatele měla být první stránkou, kterou uvidí. Obsahuje nějaké základní informace o hře a formuláře pro připojení se. Tyto formuláře existují tři – první z nich slouží pro připojování se ke hře a zobrazí se jen pokud má hráč v `localStorage` uložený token hry, který mu je přidělen v průběhu připojování se. Druhý slouží k připojení se pomocí kódu hry a jména a třetí slouží k vytvoření hry. U vytvoření hry je vybírací nabídka dostupných pluginů, která se musí nejdříve dostat od serveru, aby se mohli zobrazovat názvy her, popisy a pravidla.

12.4 Hlavní objekt `gamestate`

GameState je objekt držený v `GameKontextu`, který obsahuje všechny informace o probíhající hře. Objekt též obsahuje informaci zda je hráč připojen ke hře (`inGame`), zda je hra spuštěna (`gameStarted`) a seznam pluginů, které jsou na serveru nainstalované, včetně jejich metadat (`gameTypesAvailable`).

12.5 Herní stránka `GamePage`

Herní stránka je stránka, která se zobrazuje, když je hráč připojen ke hře a hra je spuštěna. Obsahuje všechny herní prvky, které jsou zobrazené hráči. Kromě prvků obsahuje utility drag & drop kontext `DndContext` a `<GlobalNotifications />`.

Stránka se dělí do tří částí pod sebou – panel hráčů `Players` s informacemi ke každému hráči, centrální panel `CentralPanel` s kartami na stole a dalšími prvky společné pro všechny hráče a panel s prvky související pouze s hrajícím hráčem `MyThings`.

12.6 Drag & drop

Hra používá drag & drop logiku pro přesun karet, díky kterým může hráč jedním pohybem určit jak co a kam chce předělat mnohem přirozeněji než několika kliknutími. Pro tento účel se používá knihovna *react-dnd*. Původně bylo učiněno několik pokusů o vlastní implementaci, ale nakonec se ukázalo, že pro tento projekt je lepší použít hotové řešení, které je dobře zdokumentované, podporované a obsahuje řešení všech hraničních případů i typů uživatelského vstupu.

12.7 Notifikace

Hra používá více systémů notifikací. Prní z nich jsou toast notifikace v pravém horním rohu pro informování hráče. Používají se, když se stane nějaká událost, která nemusí být jasně patrná z drobné změny UI, která proběhla. Druhým typem jsou notifikace, které se zobrazují přímo uprostřed. Cílem těchto notifikací je upozornit na sebe i když hráč například nedávvá pozor. Tyto notifikace se používají například pro upozornění na začátek tahu.

Pro toasty se používá knihovna *react-hot-toast*, která poskytuje jednoduché API pro zobrazení toastu. Knihovna byla vybrána i přes to, že by šla tato funkčnost vyrobyt bez ní, ale nebyl důvod „znovu vynalézat kolo“. Toast jde po importu vyvolat pomocí funkce `toast()`, která má několik variant pro různé typy notifikací, například `toast.error()` pro chybové notifikace.

Pro notifikace uprostřed se používá vlastní komponenta `CentralNotification`, která zobrazuje zprávu jednoduchý text, který se uprostřed obrazovky začnezvětšovat a zprůhledňovat. Komponenta využívá portál, kterým do `<body>` stránky vkládá absolutně pozicovaný element, do kterého vykresluje samotnou notifikaci. Po importu komponenty jde notifikace vyvolat metodou `notify()`.

Pokud je třeba hráče informovat tak, že musí přečtení potvrdit, tak se použije dialog.

12.8 Dialog

Dialog je komponeta se svím kontextem, která poskytuje api pro informování hráče a získávání vstupu od něj. Skádá se z `DialogProvideru`, který vyrábí kontext a poskytuje metody `openDialog` a `closeDialog`. `DialogProvider` vrací kontext `DialogContext`, který exportuje tyto metody a potřebné typy.

Samotná komponenta `Dialog` používá kontext dialogu a v případě otevření vytváří portál, ve kterém dává obal dialogu, ztmavené pozadí a samotný obsah dialogu. Ten se určí podle typu dialogu. Existují typy `SELECT_CARD`, `SELECT_PLAYER`, `CONFIRM` a `INFO`.

Dialog má různé možnosti, například zda se může zavřít bez akce, kolik položek může maximáálně i minimálně vybrat a podobně.

12.9 Tmavý režim

Celý web je vyroben spodporou tmavého režimu. Pro tmavý režim se nepoužívá `prefers-color-scheme`, aby uživatelé mohli používat přepínač v aplikaci. Stav tmavého režimu ovládá přidáváním třídy `darkMode` na `<body>` element. Nepoužívá se kontext, aby se nemusel překreslovat framework a všechny komponenty, ale pouze css interně v prohlížeči. V `.module.css` souborech se pak používá `:global(.darkMode)` pro funkčnost i po kompilaci, která by jinak změnila názvy tříd.

Přepínač je komponenta `DarkModeSwitch`, která se vykresluje jako měsíček či sluníčko v pravém horním rohu. Tento přepínač se neukazuje na normální hrací stránce, kde by byl zablokovaný, ale režim jde přepínat například při otevřeném dialogu.

12.10 Server info

Nepřímou součástí klienta je i html stránka `serverInfo.html`, která se nachází v `/public/` adresáři. Tato stránka slouží pro zobrazení informací o serveru, které jsou užitečné pro správu serveru, například pro zjištění aktuálního počtu připojených hráčů, seznamu her a podobně. Stránka získává data pomocí websocketového připojení k serveru a zobrazuje je v jednoduchém formátu, který poskytuje server.

12.11 Původní klient

V zaříjové fázi projektu se místo současného klienta v *React*u používal klient ve vanilla JavaScriptu. I přes to, že již není vyvíjen, tak některé základní části stále podporuje a proto se v repozitáři stále nachází ve složce `klient`.

13 Utility pro vývoj

Kromě hlavních částí projektu bylo během vývoje vyrobeno několik užitečných python programů, které pomáhaly s různými úkoly. Například pro generování obrázků karet byl pomocí umělé inteligence vtvořen python skript, který pomocí knihovny *PIL* generoval obrázky karet na základě zadaných parametrů. Tento skript umožnil rychle vytvořit velké množství karet pro různé pluginy bez nutnosti ručního designování každé karty. Další užitečný script stahoval karty z prší do konkrétní složky, což urychlilo proces získávání potřebných obrázků pro tento plugin.

14 Hosting a nasazení

Jak již bylo zmíněno, tak je hra hostovaná na studentském serveru Gymnázia Arabská avava. Tento server poskytuje *Caddy* server, který se při požadavku chová nejprve jako statický server pro soubory, ale jakmile nějaký soubor neexistuje, tak předá požadavek na backendový server tím, že zavolá na konkrétní port. Na tomto portu běží server projektu s websocketem. Aby se nevypnul při odhlášení se z SSH, tak se používá nástroj *tmux*, který umožňuje spouštět procesy na pozadí a připojovat se k nim později. Na statické části serveru je uložen i předem postavený klient, který se načítá hráčům, když navštíví webovou stránku. Pro připojení k websocketu se používá `/ws` endpoint.

15 Testování

Během vývoje byla hra nesčetněkrát testována jejím autorem. Kromě toho se mnohokrát testovala i s jeho kamarády a rodinou, kteří hru zkoušeli a upozorňovali na nedostatky. Hra se testovala jak na lokálním serveru, tak i na veřejném serveru, aby se otestovalo chování v různých sítích a s různými počty hráčů.

Díky tomu, že se do enginu přidaly rychle jednoužší hry, tak bylo testování zábavné a proto se mohlo testovat často a s různými lidmi, což pomohlo odhalit spoustu nedostatků a zlepšit hru.

Pro testování existuje i několik nástrojů, například pro zobrazení boxů kolem všech prvků DOM stačí přidat body třídu `testovani`, nebo po zobrazení celé websocketové komunikace stačí otevřít konzoli prohlížeče a podívat se na zelené zprávy.

16 Budoucí rozšiřování

Projekt je navržený tak, aby se dal snadno rozšiřovat. Díky pluginům je možné přidat další hry, které budou fungovat na stejném enginu. V plánu je přidat některá rozšíření pro *Bang!*, které již jsou předpřipravené. Dále by bylo možné přidat i další funkce pro klienta, například možnost přizpůsobení vzhledu, přidání zvuků a podobně.

17 Závěr

18 Přílohy

19 Zdroje

Reference

[1] Dan Abramov. React dnd: Drag and drop for react. https://react-dnd.github.io/react-dnd/about.

[2] Baeldung. A guide to org.json. https://www.baeldung.com/java-org-json.

[3] Baeldung. List files in a directory in java. https://www.baeldung.com/java-list-directory-files.

[4] Claudéric Demers. dnd kit: A modern drag and drop toolkit for react. https://github.com/clauderic/dnd-kit.

[5] Ecma International. The json data interchange syntax (standard ecma-404). https://ecma-international.org/publications-and-standards/standards/ecma-404/.

[6] Dark gray wooden plank pattern. https://www.freepik.com/free-photo/dark-gray-wooden-plank-pattern_1037175.htm.

[7] Google. Gemini ai, 2024-2026. Analýza chybových hlášení, optimalizace dokumentace a konzultace Maven příkazů.

[8] Google Inc. Gson api documentation. https://www.javadoc.io/doc/com.google.code.gson/gson/latest/com.google.gson/com/google/gson/Gson.html.

[9] Interní systém avava. https://avava.gyarab.cz/.

[10] Průvodce svs: Statický web. https://guides.svs.gyarab.cz/staticky-web.html.

[11] ITnetwork. Latex tutoriál: Nadpisy a struktura. https://www.itnetwork.cz/programovani/latex/latex-tutorial-nadpisy.

[12] Představení json. https://www.json.org/json-cz.html.

[13] Timo Lins. React hot toast: Smoking hot notifications. https://react-hot-toast.com/.

[14] Meta Platforms, Inc. React reference: usecontext. https://react.dev/reference/react/useContext, 2024.

[15] Meta Platforms, Inc. React reference: useeffect. https://react.dev/reference/react/useEffect, 2024.

[16] Meta Platforms, Inc. Rendering lists - keeping list items in order with key. https://react.dev/learn/rendering-lists#keeping-list-items-in-order-with-key, 2024.

[17] Microsoft. Github copilot, 2024-2026. Asistent při psaní kódu v TypeScriptu a Javě.

[18] Bang! the bullet product page. https://www.m-g.com.au/product/bang-the-bullet/.

[19] Mozilla Contributors. Html drag and drop api. https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API.

[20] Mozilla Contributors. Json.parse: Bad parse errors. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Errors/JSON_bad_parse.

[21] OpenAI. Chatgpt (verze gpt-4o a gpt-o1), 2024-2026. Použito pro generování Python skriptů pro zpracování obrázků a konzultace herní logiky.

[22] Oracle Corporation. How to write doc comments for the javadoc tool. https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html.

[23] Martin Pulido. How to make a bang! character card. https://bangcardgame.blogspot.com/2011/03/design-how-to-make-bang-character-card.html.

[24] Is there a way to create virtual ram in ubuntu? https://www.reddit.com/r/Ubuntu/comments/1gqneha/is_there_a_way_to_create_virtual_ram/.

[25] Print and play version for bang! https://www.reddit.com/r/printandplay/comments/6p3ar6/print_and_play_version_for_bang/.

[26] Do the json keys have to be surrounded by quotes? https://stackoverflow.com/questions/949449/do-the-json-keys-have-to-be-surrounded-by-quotes.

[27] Java: Convert integer to string. https://stackoverflow.com/questions/5071040/java-convert-integer-to-string.

[28] Java: Leaking "this"in constructor. https://stackoverflow.com/questions/9851813/java-leaking-this-in-constructor.

[29] Random shuffling of an array. https://stackoverflow.com/questions/1519736/random-shuffling-of-an-array.

[30] Rotate an image in html. https://stackoverflow.com/questions/20061774/rotate-an-image-in-image-source-in-html.

[31] List subdirectories only n-level deep. https://unix.stackexchange.com/questions/93323/list-subdirectories-only-n-level-deep.

[32] Vite: Building for production. https://vite.dev/guide/build.

[33] Bang! (karetní hra). https://cs.wikipedia.org/wiki/Bang!_(karetn%C3%AD_hra), 27. 11. 2025, 02:20.