Gymnázium, Praha 6, Arabská 14

Obor Programování

Ročníková práce

Mariáš

Jakub Turek

Duben 2022

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 Brandýse nad Labem dne Jakub Turek

Anotace

Následující práce pojednává o tvorbě aplikace založené na modelu klient-server v programovacím jazyce Java, která slouží pro hraní karetní hry mariáš na několika počítačích nacházejících se v jedné lokální počítačové síti. Text se dále zabývá také prací s externí knihovnou JavaFX, která slouží pro tvoru grafických rozhraní v jazyce Java a je použita pro tvorbu grafického rozhraní klientského programu.

Abstract

The main topic of following thesis is the creation of a graphics application in Java language, which would be used to play card game called mariáš at multiple devices in local network. The secondary topic of this thesis is using external Java library JavaFX, which is used for creation GUI of Java applications and it is also used for creation of this project client program GUI

Obsah

Úvod

Tématem této práce je karetní hra mariáš, jejíž pravidla budou podrobněji uvedena v následující kapitole, konkrétně tvorba grafické aplikace, která by umožnila několika hráčům tuto hru hrát na více zařízeních a nejednalo by se tedy o lokální multiplayer.

Tato aplikace se skládá ze dvou částí a to jednoduché serverové části, kterou spustí jeden z hráčů na svém zařízení. Tato část aplikace vyčká na připojení čtyř klientských programu a na základě jimi odeslaných dat, které zároveň posílá zbylým klientům, vyhodnocuje jednotlivé herní aspekty a tyto výsledky vrací klientům. Klientské programy, jak lze vyvodit z již zmíněných skutečností, neobsahují žádnou logiku, která by se týkala samotné hry a starají se pouze o korektní zobrazení všech nezbytných údajů hráči, prostřednictvím grafického rozhraní vytvořeného pomocí knihovny JavaFX a kaskádových stylů, a o samotnou komunikaci se serverem.

I. Pravidla hry a řešená problematika

Tato kapitola popisuje pravidla karetní hry mariáš a následně nastiňuje z nich vyplývající problematiku, kterou je třeba řešit při vytvářené aplikace pro její hraní.

I.I. Pravidla hry

Hra mariáš mám mnoho variant, které mají rozdílná pravidla, aplikace se proto zaměřuje pouze na klasickou variantu pro čtyři hráče.

Všechny varianty mariáše se hrají s kartami německého typu, které jsou někdy nazývány jako "mariášky". Sada karet je tvořena čtyřmi barevnými řadami karet srdce, listy, žaludy a kule a každá řada je tvořena osmy kartami hodnoty sedm až deset, spodek, svršek, král a eso, toto pořadí určuje jejich hodnotu ve hře. Každý hráč tak na začátku partie obdrží osm náhodných karet, se kterými hraje po zbytek partie. Pokaždé, když se hráč dostane na tah, musí zahrát jednu ze svých karet a každá partie trvá dokud se všichni hráči nezbaví svých karet a má tak tedy vždy přesně osm kol.

Na začátku každé partie po rozdání karet zvolí začínající hráč trumfovou barvu pro tuto partii. Karty trumfové barvy mají vyšší hodnotu než ostatní karty ve hře, ale dají se zahrát jen za určitých podmínek, které budou popsané později. Zároveň zvolí také kartu, tedy barvu a hodnotu například srdcová deset, tímto výběrem karty si hráč určí spoluhráče po tuto partii, stane se jím hráč, který tuto kartu drží, kdo s kým hraje tak přesně ví jen držitel vybrané karty. Zbylí dva hráči hrají spolu a tvoří tak druhý tým. Na konci partie vítězí tým, jehož členové získali více bodů.

Poté co jsou provedeny tyto úvodní akce může začínající hráč vynést libovolnou kartu ze své ruky a poté jeho tah končí a hraje další hráč. Další hráč provede stejnou akcí s tím rozdílem, že už nemůže zahrát libovolnou kartu, ale musí zahrát kartu stejné barvy a vyšší hodnoty nežli karta hraná hráčem předním. V případě, že hráč nemá kartu vyšší hodnoty, musí alespoň respektovat barvu předchozí karty. Pokud nemá hráč kartu vyšší hodnoty ani požadované barvy musí požít libovolnou kartu trumfové barvy. Pokud hráč nemá ani trumfovou kartu muže zahrát libovolnou kartu ze své ruky. Tuto akci postupně provedou všichni hráči a po odehrání posledního z nich dojde k vyhodnocení kola.

Jak již bylo řečeno trumfové karty mají vyšší hodnotu než ostatní karty ve hře a pokud tedy jsou v daném kole zahrány trumfové karty vítězí v něm hráč, který zahrál nejvyšší trumfovou kartu. V případě, že nikdo trumfovou kartu nezahrál vítězí hráč, který zahrál nejvyšší kartu barvy vynesené začínajícím hráčem daného kola. Vítěznému hráči se připočtou body. Za každou desítku a eso zahrané v daném kole obdrží hráč po deseti bodech. V případě, že má hráč v ruce dvojici král a svršek stejné barvy, jedná se o takzvanou "hlášku" a ve chvíli kdy hráč hraje jednu z karet obdrží dvacet bodů (body obdrží hráč hrající hlášku nikoliv vítěz kola ve kterém byla zahrána). Hra v tomto ohledu obsahuje ještě celou řadu modifikátoru měnících výše výchozích peněžitých sázek. Avšak vzhledem k tomu, že jsem se rozhodl systém sázek do svého projektu nezapracovat nebudu je zde zmiňovat.

Po vyhodnocení kolo začíná kolo nové s tím, že vynáší hráč hrající v minulém kole jako druhý. Tento postup se opakuje dokud není odehráno všech osm kol. Vyhodnocení posledního kola se tým s vyšším počtem bodů stává vítězným.

I.II. Problematika

Z uvedeného zadání a popisu pravidel hry, kterou aplikace umožňuje hrát, již vyplývá problematika řešená v rámci této práce. Této kapitola tuto problematiku ve stručnosti nastíní.

Prvním z problému, který je třeba v projektu vyřešit a vyplývá hned z úvodního zadání je navázání komunikace mezi jednotlivými hráči a umožnit jim tak sdílet informace nezbytné ke hře.

Poté co je již navázáno spojení mezi a dochází ke sdílení informací je nutné zajistit řešení hlavní úlohy za nímž celá aplikace hry vzniká a to hraní karetní hry mariáš. Podúlohy řešené v rámci tohoto problému lze podrobněji najít předchozí sekci popisující pravidla hry, zkráceně se ale jedna o vytvoření balíčku karet, rozdání karet, přijetí výchozích údajů od hráčů (trumfová barva, karta s níž hráč hraje atd.), reakce na zahrané karty, vyhodnocování kol a s tím spojené bodování hry a závěrečné vyhodnocení celé hry.

Poslední z větších úloh na kterou by se řešená problematika dá rozdělit je komunikace s hráčem prostřednictvím uživatelského rozhraní. V rámci tohoto problému je třeba řešit zobrazování hracích karet držených hráčem, hraní těchto karet stylem

drag and drop, tedy uchopením karty kurzorem a jejím přetažením na určité místo, zobrazování karet zahraných jinými hráči a zobrazovaní informací o nich (jméno, body atd.).

Toto by tedy bylo krátké nastínění problematiky řešené v tomto projektu a nyní již budou následovat kapitoly zabývající se samotným návrhem aplikace a řešením výše popsané problematiky, respektive jejích významných podproblémů.

II. Návrh

Tato kapitola pojednává o návrhu aplikace. První podkapitol se zabývá strukturou samotného kódu aplikace. Následující kapitola pojednává o rozdělení aplikace z pohledu uživatele a řešení těchto částí, tedy návrhem významných tříd a tříd které složí jako datové struktury ukládající data hry.

II.I. Struktura kódu

Samotná aplikace je pro vetší přehlednost kódu rozdělena do třech samostatných balíčku obsahujících kód a jedné složky obsahující obrázky pro grafické uživatelské rozhraní klienta. Tyto balíčky budou popsány níže.

II.I.I. Klient

Tento balíček obsahuje kód využívaný pouze klientským programem. Jelikož je samotný klientský program navržen na modelu MVC (model, view, controller) obsahuje tento balíček jednak soubory typu .fxml ve kterých je definováno samotné rozvržení jednotlivých prvku GUI. Dále obsahuje třídy které složí jako řadiče (controller), které spravují změny GUI za běhu aplikace. Dále obsahuje ještě klasické třídy, které obsahují logiku pro dané okno a zároveň odkaz na jeho řadič. Dále obsahuje ještě třídy reprezentující zbylou logiku klienta.

II.I.II. Sdílené

Balíček sdílené jak již název napovídá obsahuje prostředky sdílené jak klientským tak serverovým programem. Tento balíček tak obsahuje třídy sloužících k reprezentaci dat, která jsou odesílána mezi serverem a klientem, jako jsou třídy *ServerParametr* reprezentující parametry serveru jako port, název a adresa na níž běží, *Card* reprezentující jednotlivé karty a *CardManager* údaje o balíčku karet. Kromě těchto tříd obsahuje také dvě třídy obsahující logiku požívanou jak serverem tak klientem, konkrétně se jedná o třídy obstarávající příjem a odesílání dat.

II.I.III. Server

Tento balíček obsahuje třídy serverového programu. Jelikož nemá serverová část vlastní GUI a uživatel jí spouští skrze uživatelské rozhraní klientského programu, obsahuje balíček pouze soubory typu .java. Konkrétně třídu Game, která je potomkem třídy Thread a reprezentuje hlavní vlákno serveru, a ještě několik tříd obsahujících další logiku serveru.

II.II. Části aplikace a jejich provedení

Tato podkapitola se zabývá členěním aplikace z pohledu uživatele tedy funkčnosti a technické provedení těchto části, což znamená popis funkce jednotlivých tříd a jejich vztahu s ostatními a popis tříd reprezentujících data.

Jak již bylo zmíněno výše aplikace je z tohoto hlediska rozdělena na serverový a klientský program. Řešení těchto částí pude podrobněji popsáno v následujících podkapitolách.

II.II.I. Klientský program

Klientský jak už je u tohoto typu aplikací zvykem neobsahuje téměř žádnou logiku a stará se hlavně o obsluhování událostí, které vyvolává uživatel prostřednictvím GUI a zároveň přijímá data od serveru a na jejich základě aktualizuje GUI.

Co se týče grafického rozhraní a jeho obsluhy, tak samotné rozhraní se skládá ze dvou hlavních oken (Úvodní obrazovka a samotná herní obrazovka) a čtyř dialogových oken (výběr existujícího serveru, vytvoření serveru a dvě pro zadávání trumfové karty a spoluhráče)[3]. Všechna tato okna jsou definována v samostatném .fxml souboru. Každé okno k sobě má ještě přiřazený jeden řadič, který obsluhuje události vyvolávané uživatelem v okně a také obstarává jeho aktualizaci.

Přestože jsou všechna okna definováná .fxml souborem, jsem se rozhodl pro přidávání karet do rozložení herní obrazovky z kódu jejího řadiče pomocí cyklu, jelikož manuální definice .fxml by byla pro jejich počet neefektivní. Pro grafickou reprezentaci karet jsem vytvořil vlastní JavaFx prvek reprezentovaný třídou *ImageViewWhitCordinates*, která dědí od výchozí JavaFX třídy *ImageView*. Oproti své rodičovské třídě má třída čtyři atributy typu double pro uložení souřadnic své

výchozí pozice na obrazovce a pro uložení souřadnic kurzoru během toho co je jím popotahovaná po obrazovce a také jeden atribut typu boolean, který je před každým tahem hráče aktualizován a udává zdali je karta v daném tahu hratelná. Atributy souřadnic slouží pro snazší implementaci darg and drop mechaniky[8][2] na karty a atribut hratelnosti zabraňuje hráči hrát nevalidní karty (v případě jejich zahrání se vrátí na výchozí pozici a klient neodešle žádná data).

V případě dialogového okna pro připojení k existujícímu serveru je okno reprezentováno ještě jednou třídou obsahující logiku potřebnou pro jeho obsluhu.

Okno pro připojení k existující hře je reprezentováno Třídou *ExistingGameScreen*, jejíž instance obsahuje odkaz na řadič dialogového okna a tak muže zpracovávat přijatá data o serverech v lokální síti a výsledky předávat jako parametry metodám volaným na řadiči, které se pouze postarají o jejich korektní zobrazení.

V případě, že uživatel vybere jeden z nalezených serveru a rozhodne se k němu přípojit, dojde ke spuštění části aplikace, která reprezentuje a zajišťuje samotné propojení se serverovou částí. Tato část programu je reprezentována třídou *Client*, která je jednak potomkem třídy Thread, což znamená že muže být spuštěna jako samostatné vlákno a ve smyčce přijímat a případně odesílat data serveru a reagovat na nimi vyvolaný stav aplikace. A zároveň je navržena podle návrhového vzoru jedináček[1], což umožňuje v zánik pouze jedné její instance, která je dostupná napříč programem za účelem dostupnosti jí reprezentovaného spojení ve všech jeho částech. Toto řešení není úplně optimální, ale pro účely tohoto projektu je dostačující a snadné na implementaci.

V tomto ohledu aktualizace GUI je ještě zajímavé zmínit problém, na který jsem narazil a to že v případě JavaFX nelze volat požadavky na aktualizaci GUI z jiného nežli hlavního vlákna aplikace. Pokud chcete aktualizovat GUI z jiného vlákna je nutné požít třídu *Platform* knihovny a zavolat na ní metodu *runLater()* s parametrem, který implementuje *Executable* a v metodě run() obsahu kód pro aktualizaci. Nebo požít lambda výraz jako na obrázku níže[12].

```
// Zobrazí uživateli dialogová okna z vlákna pro příjem dat.
 Platform.runLater(
        () -> {
            try {
                screenManager.showDialog("trumphDialog.fxml");
                screenManager.showDialog("playWithDialog.fxml");
            } catch (IOException ex) {
                screenManager.showExceptio(ex);
            }
        }
 );
```

Příloha 1: Otevření dialogu s užitím *Platform.runLater()* a lambda výrazu

Nyní již zpět k třídě *Client*. Instance třídy reprezentuje jednoho klienta připojeného k serveru. Poté co se uživatel rozhodne připojit je vytvořena instance této třídy s parametry potřebnými pro vytvoření instance třídy *Socket*, která naváže spojením se vybraným serverem. Když je server zaplněn dojde k *inializaci* hry, tedy server přijme od všech hráčů jejich jména uloží si je a zároveň je odešle zbylým hráčům, aby mohla být zobrazena. Zároveň server hráčům odešle jejich hrací karty, které jsou reprezentovaný jako *ArrayList* instancí třídy *Card* popsané výše.

Po dokončení této inicializace se na serveru spustí herní smyčka, popsaná v další kapitole, zatímco instance třídy *Client* začne ve smyčce přijímat dat ze serveru a dle jejich typu volá metody , které data dále zpracovávají. Po odehrání osmy kol se obě smyčky ukončí a server odešle údaj, kdo ve hře zvítězil.

II.II.II. Serverový program

Serverový program nemá vlastní uživatelské rozhraní a je spuštěn jedním z hráčů na pozadí poté co vytvoří novou hru. Právě vzhledem k absenci GUI je serverový program, i přestože obsluhuje veškerou herní logiku jednoduší než klient.

Server aplikace je reprezentován třídou Game jejíž instance se vytvoří s parametry zadanými uživatel v GUI klienta. V rámci instance Game se vytvoří také instance třídy *ServerSocket* se kterým později naváží spojení klienti. Třída Game je potomkem třídy Thread což umožňuje její instanci spustit jako vlákno na pozadí bez toho aniž by hráč zpozoroval jeho běh.

Poté co je instance spuštěna a připojí se všichni hráči, proběhne inicializační sekvence hry popsaná již v části o klientovi. Zde bych jen dodal, že instance Game

v sobě ukládá veškeré informace o připojených hráčích a server je tak připraven na přidání funkce na znovupřipojení klienta po ztrátě spojení, kdy mu jednoduše odešle uložená data. Data o hráčích jsou na server ukládá v podobě pole instancí třídy *Player*, která reprezentuje jednoho hráče. Třída *Player* je jednoduchá tvoří jí jen atributy pro uložení jména jako *Stringu*, bodů jako intu a množiny karet jako *ArrayListu* objektů typu *Card* a gettrů a settru pro nastavování a získávání jejich hodnot.

Poté se již spustí herní smyčka. Vzhledem k tomu, že charakter hry vyžaduje aby hráli hráči v pevně daném pořadí, z tohoto důvodu je ideálním řešením požít pro server jen jedno vlákno a všechny klienty obsluhovat postupně. Proto jsou klienti připojení k serveru uloženi v jednoduchém poli o čtyřech prvcích, které je postupně v cyklu procházeno. Server nejprve danému klientovi pošle pošle objekt typu boolean, který značí, že daný hráč je na tahu. Poté klient aktivuje své hratelné karty pro daný tah. Serve následně přijme objekt typu *Card* reprezentující zahranou kartu. Tuto kartu si uloží a odstraní jí z množiny karet uložené pro daného hráče na serveru a odešle jí zbylým hráčům. Tato procedura se opakuje dokud nejsou obslouženi všichni klienti. Po obsloužení všech klientů vyhodnotí server odehrané kolo. O vyhodnocování jednotlivých herních aspektů se stará třída *GameManager* na jejíž instanci se z instance Game volají patřičné metody. Výsledky vyhodnocení opět odešle server klientům. Herní smyčka se opakuje dokud není odehráno všech osm kol kdy je opět pomocí *GameManageru* vyhodnocena celá hra a výsledek je odeslán.

Poslední třídou serveru je *ServerInfoSender*, která je slouží k odesílání informací o serveru tedy ip adresu zařízení na kterém běží v lokální síti, název zvolený jeho tvůrcem a port na kterém poslouchá. Tyto informace jsou pak potenciálním klientům zobrazeny na obrazovce existujících her. Tato třída společně s třídou *ExistingGamesFinder* klienta tvoří mechanismus vyhledávání již založených her, který bude popsán v další kapitole a dělá tak připojení ke hře velmi snadné.

III. Hlavní problémy a jejich řešení

Následující kapitola se zaměřuje na přiblížení hlavních problémy problému, které bylo třeba během vývoje vyřešit a jejich řešení.

III.I. Vyhledávání již existujících her

Ač to přímo nesouvisí s hlavním problémem to jest hraní karetní hry mariáš, tak hraní hry na více zařízeních tento problém nutně implikuje a to najít způsob, jak zviditelnit již existující hry založené v síti ostatním hráči a umožnit tak hráčům pohodlné připojování k jednotlivým hrám, bez toho aby museli nějakým způsobem vědomě sdílet například svou ip adresu a port na kterém server poslouchá.

Řešení tohoto problému nakonec nebylo až tak technicky náročným, jak by se mohlo na první pohled zdát. Pro řešení tohoto problému se ukázalo jako nelepší využít vysílání typu broadcast založené na UDP protokolu[9][7], který sice neručí za doručení dat, ale oproti TCP protokolu který je používán pro komunikaci mezi serverem a klientem vyžaduje méně prostředků na režii spojení a komunikace jeho prostřednictvím je jednoduší[11], což jsou pro toto použití žádoucí vlastnosti.

Při vytvoření instance třídy Game se proto vytvoří i instance třídy *ServerInfoSender*, která se spustí jako samostatné vlákno a začne ve smyčce ve dvou vteřinových intervalech odesílat základní informace o serveru na broadcastovou
adresu 255.255.255.255 což je broadcastová adresa lokální sítě[7]. Port který je pro komunikaci požíván je ve výchozím nastavení nastaven na 49152, ale uživatel ho může změnit v konfiguračním souboru config.txt (pro korektní fungování ho však musí změnit i ostatní hráči kteří se chtějí účastnit stejné hry, proto doporučuji výchozí port neměnit, pokud to není nezbytně nutné).

Při vysílání typu broadcast na broadcastovou adresu sítě dochází k doručování dat všem uzlům dané podsítě, a proto se dostanou ke všem potencionálním hráčům.

Pro fungování celého vyhledávání je kromě vysílání na straně serveru, ale také příjem a zpracovaní dat na straně klienta. Za tímto učelem disponuje klientský program třídou *ExistingGameFinder*, který v nekonečné smyčce přijímá data na základě komunikace prostřednictvím UDP na výchozím portu. Z takto přijatých dat ve formě textového řetězce vytváří objekty typu *ServerParametr* jejichž seznam

následně předá jako paramet metodě controlleru okna existujících her, která se stará o zobrazení seznamu aktivních her a je volaná každých deset sekund, aby docházelo k automatické aktualizaci.

III.II. Vyhodnocování her

Dalším významným problémem, který již souvisí se samotným hraním hry je vyhodnocování jednotlivých kol hry a posléze samotné hry.

Tento problém není příliš náročný na řešení, jelikož jak je popsáno v kapitole pravidla je pro vyhodnocování kola nutné kontrolovat jen jednu podmínku, a to zdali se mezi zahranými kartami nachází karty trumfové barvy. Pokud ano, tak vítězí nejvyšší trumfová karta. Pokud ne, tak vítězí karta nejvyšší hodnoty, která má stejnou barvu jako první karta zahraná v daném kole.

Jelikož jak klienti, tak jejich data jsou uložena serverem pomocí seznamu objektu, jsou i karty uloženy do stejně indexovaného seznamu a tak stačí pouze získat index vítězné karty podle popsáného schématu a získáme i index na němž jsou uložená data vítězného hráče a ty tak mohou být aktualizována a následně odeslána všem klientů pro zobrazení.

```
/*
Vrátí index vítězné karty, která je zároveň indexem
dat o vítězném hráči
*/
public int getWinerOfRound(List<Card> playedCards, int indexOfFirst) {

    if (containsTrump(playedCards)) {

        return indexOfHigestValue(cardsFilter(playedCards, TRUMPHCOLOR));

    }

    return indexOfHigestValue
    (cardsFilter(playedCards, playedCards.get(indexOfFirst).getColor()));

}
```

Příloha 2: Metoda vyhodnocující jednotlivá kola

Samotné vyhodnocování celé hry je ještě jednoduší, jelikož server má uložené množiny karet, stačí aby našel hráče, který drží kartu se kterou hraje první hráč

a poté na konci hry porovnal body získané oběma týmy a za vítěze označit ten s výším bodovým ziskem.

Vykreslování karet

Posledním z významnějších problému bylo zobrazování příslušných karet na základě dat přijatých od serveru.

GUI reprezentuje zahrané karty jako klasická *ImageView* z knihovny JavaFX, která zobrazují určitý obrázek formátu PNG symbolizující jím reprezentovanou kartu a karty které má hráč k dispozici zase jako *ImageViewWithCordinates*, která byla popsána výše a též symbolizuji reprezentovanou kartu pomocí PNG obrázku.

Vzhledem k této reprezentaci jsem původně uvažoval o užití jednoduché hašovací tabulky, která by klíčům v podobě názvu karty (barva + hodnota) přiřazovala adresy na je reprezentující obrázky.

Toto řešení jsem se ale následně rozhodl nepoužít pro malou efektivitu a nutnost vytvářet pole, které by obsahovalo 32 adres. Z tohoto důvodu jsem se rozhodl, že metody *toString()* instance *Card* bude vracet textový řetězec, kterému bude odpovídat název souboru zobrazující kartu dané hodnoty a barvy.

Toto řešení tedy pouze vezme textový řetězec a s pomocí následující metody z něj získá *InputStream* obrázku reprezentující danou kartu. Z důvodu následného zabalení souborů do spustitelného souboru nelze použít načtení pomocí cesty k souboru reprezentované textovým řetězcem, ale je nutné použít toto řešení[5][6].

```
/*
    Získá obrázky jako InputStream a předá je controlleru
    pro vykreslení
*/
  public void dealCards(List<Card> cards) {
        InputStream[] imagesStreams = new InputStream[cards.size()];
        for (int i = 0; i < cards.size(); i++) {
            imagesURLs[i] = getClass().getResourceAsStream
            ("/pictures/" + cards.get(i).toString() + ".png");

        }
        controller.dealCards(imagesStreams);
    }
```

Příloha 3: Metoda získávající InputStreamy obrázku reprezentujících dané karty

IV. Použité technologie

IV.I. JavaFX

Framework pro tvorbu uživatelského rozhraní v Javě. Umožňuje tvorbu jak desktopových, mobilních i webových aplikací. Pro tvorbu uživatelského rozhraní využívá značkovací jazyk FXML, který lze stylizovat s pomocí kaskádových stylů[10].

IV.II. SceneBuilder

Nastroj usnadňující tvorbu uživatelského rozhraní v JavaFX. Umožňuje navrhnout uživatelské rozhraní pomocí grafického návrháře a odpadá tedy nutnost psát FXML kód, který za uživatele vygeneruje nástroj. Uživateli poté stačí kód mírně, či vůbec manuálně editovat.

Launch4J

Nástroj pro tvorbu spustitelných souborů .exe ze souborů typu .jar. Nastroj umožňuje .jar soubor zabalit do .exe souboru a přidat mu závislosti na všechny nutné externí Java knihovny[4]. To umožňuje program distribuovat v podobě .zip archívu, který obsahuje vše potřebné pro běh aplikace a uživateli jej stačí jen rozbalit na požadovaném místě a spustit v něm se nacházející spustitelný soubor, což činí jeho distribuci velmi pohodlnou.

V. Instalace

Instalace aplikace je uživatelsky velmi přívětivá. Jelikož je aplikace napsaná v jazyce Java je nutné aby se na zařízení na němž má byt aplikaci provozována nacházelo jeho běhové prostředí. Jeho instalace je však také velmi jednoduchá stačí pouze navštívit web společnosti Oracle a zde stáhnout instalační soubor Java (verze 11 a vyšší) a ten následně spustit a projí instalací. V případě, že již máte na vašem zařízení běhové prostředí nainstalované stačí z GitHub repositáře s tímto projektem stáhnout složku dist rozbalit jí a spustit v ní se nacházející soubor marias.exe.

VI. Závěr

Závěrem bych přešel ke konečnému zhodnocení celého projektu. Již na začátek bych podotkl, že celý projekt hodnotím jako úspěšný, jelikož během jeho vývoje dosáhnout všech cílů, které byly stanoveny v zadání.

Přestože projekt splňuje všechna kritéria zadání, je zde i aspekty, který vyhovují zadání, ale stálé vyžaduje menší vylepšení. Tímto aspektem je grafické zpracovaní aplikace. Design celého GUI aplikace je značně jednoduchý, což dle mého názoru není na škodu a v tomto ohledu není nutno provádět žádná vylepšení. Vylepšení však vyžaduje hlavní herní obrazovka, kde by bylo vhodné přidat animace hraní karet soupeři a celkově přepracovat design karet.

Úprava GUI nebude nijak náročná, ale po jeho upravení bude aplikace pro uživatele více přívětivá. Z tohoto důvodu bych se zde stručně zmínil o dalším možném vývoji aplikace.

Do budoucna bych chtěl projekt případně rozšířit o další herní režimy jako je například "betl", kde hrají tři hráči proti jednomu a samostatně hrající hráč má za cíl nezískat ani jeden bod a ostatní se mu v to snaží zabránit, nebo "durch", kde opět hrají tři hráči proti jednomu, ale samostatně hrající se zde snaží vyhrát všechny bodové karty.

Touto vizí do případné budoucnosti projektu bych uzavřel jeho závěrečné zhodnocení.

Reference

[1] David Čápka. *Lekce 2 - Singleton (jedináček)*. https://www.itnetwork.cz/navrh/navrhove-vzory/gof/singleton-navrhovy-vzor/. citováno 27.4.2022.

[2] *correct way to move a node by dragging in javafx 2?* https://stackoverflow.com/questions/10682107/correct-way-to-move-a-node-by-dragging-in-javafx-2. citováno 27.4.2022.

[3] *Creating Custom Dialog*. https://riptutorial.com/javafx/example/28033/creating-custom-dialog. citováno 27.4.2022.

[4] *Creating exe files for Java Apps using Launch4j*. https://javatidbits.wordpress.com/2016/04/21/creating-exe-files-for-java-apps-using-launch4j/. citováno 27.4.2022.

[5] *How to bundle images in jar file*. https://stackoverflow.com/questions/2273040/how-to-bundle-images-in-jar-file. citováno 27.4.2022.

[6] *How to Make a JAR File with Resources (Images) - Java Extra 8*. https://www.youtube.com/watch?v=d02PK8C5EaA. citováno 27.4.2022.

[7] Deep Jain. *Broadcasting and Multicasting in Java*. https://www.baeldung.com/java-broadcast-multicast. citováno 26.4.2022.

[8] *JavaFX Drag and Drop*. https://jenkov.com/tutorials/javafx/drag-and-drop.html. citováno 27.4.2022.

[9] Petr Štechmüller. *Java server - Propagace lokální sítí (1.,2. a 3. část)*. https://www.itnetwork.cz/java/server. citováno 26.4.2022.

[10] Petr Štechmüller. *Lekce 1 - Úvod do JavaFX*. https://www.itnetwork.cz/java/javafx/uvod-do-javafx. citováno 27.4.2022.

[11] *User Datagram Protocol*. https://cs.wikipedia.org/wiki/User_Datagram_Protocol. citováno 26.4.2022.

[12] *Why am I getting java.lang.IllegalStateException "Not on FX application thread" on JavaFX?* https://stackoverflow.com/questions/17850191/why-am-i-getting-java-lang-illegalstateexception-not-on-fx-application-thread. citováno 27.4.2022.

Seznam příloh