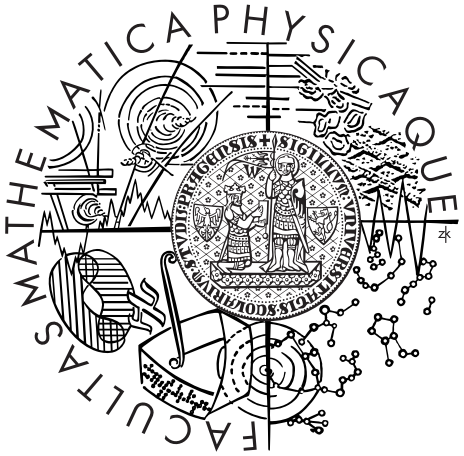
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Ondřej Papík

Generátor karetních her

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha ROK 2014

Děkuji RNDr. Janu Kofroňovi za vedení mé bakalářské práce a za užitečné rady při jejím vypracování. Dále bych chtěl poděkovat rodině za jejich podporu a pomoc při odhalování pravopisných chyb.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Generátor karetních her

Autor: Ondřej Papík

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: RNDr. Jan Kofroň, Ph.D.

Abstrakt: Mnoho karetních her momentálně nelze hrát na počítači. Cílem práce je navrhnout a implementovat program pro vytváření karetních her. Součástí práce je navrhnout programovací jazyk pro zadávání pravidel jednotlivých her. Program bude umožňovat síťové hraní a bude obsahovat i umělou inteligenci. Program bude platformě nezávislý.

Klíčová slova: obecná karetní hra, minimax, síť, překladač

Title: Card games generator

Author: Ondřej Papík

Department: Department of Distributed and Dependable Systems

Supervisor: RNDr. Jan Kofroň, Ph.D.

Abstract: Many card games can not be played on a PC at the moment. The goal of this thesis is to design and implement program for creating card games. Part of the work is to design a program language for entry rules of the games. Program is allowing network playing and is also containing artificial intelligence. Program is platform independent.

Keywords: generic card game, minimax, network, compiler

Obsah

Úvod

Existuje mnoho způsobů jak karetní hry dělit. První možností je dělení na klasické a moderní. Moderní karetní hry využívají svých vlastních karet, které se často nedají použít na jinou hru. Mezi asi nejznámější patří „Bang!“ nebo „Magic: The Gathering“. Naproti tomu klasické karetní hry se hrají s žolíkovými nebo mariášovými kartami. Ty nejsou specifické pro žádnou konkrétní hru a umožňují hrát nepřeberné množství her a variací. Další dělení může být podle počtu hráčů. Určitě je rozdíl mezi hrami pro jednoho hráče a pro více hráčů.

Další způsob, jak můžeme karetní hry rozdělit, je na tahové karetní hry, tedy takové, kdy se jednotliví hráči střídají ve svých akcích, a hry kdy všichni hráči hrají najednou. U netahových her často jde více o postřeh a rychlost, než o přemýšlení, proto se pro jednoduchost budeme zabývat pouze tahovými karetními hrami.

V dnešní době máme již mnoho karetních her naprogramováno na počítači. To umožňuje hráčům zahrát si svou oblíbenou hru proti mnoha různým hráčům po síti nebo pokud nemá tuto možnost, tak si ji zahrát proti umělé inteligenci. I když počet takto naprogramovaných her roste, tak stále ještě nejsou naprogramovány všechny. Také se neustále vymýšlejí nové karetní hry, popřípadně variace již existujících her.

Proto se pokusíme vytvořit nástroj, který pomůže uživatelům vytvářet jejich vlastní karetní hry. V první fázi navrhneme systém pravidel. Jeho správný návrh je důležitý, aby uživatel měl co nejméně práce, ale zároveň byl co nejvíce univerzální a umožňoval zapojení umělé inteligence. Následně tento návrh naimplementujeme.

V první kapitole si popíšeme co vše od našeho programu očekáváme, některá již existující řešení a popíšeme základy našeho nového řešení. Ve druhé kapitole si rozebereme strukturu programu. Ve třetí kapitole se budeme zabývat popisem hlavních algoritmů programu. Ve čtvrté kapitole se nachází programátorská dokumentace s popisem implementace. V závěru práce zhodnotíme naše výsledky.

1. Analýza problému

V této kapitole si nejdříve shrneme, čeho všeho chceme dosáhnout. Potom se podíváme na některá již existující řešení, a nakonec navrhneme své vlastní.

1.1 Požadavky

Všech karetních her je příliš velké množství. Tedy se nám zřejmě nepovede vytvořit nástroj, který umožní hraní absolutně všech her. To je nemožné i z toho důvodu, že někdo může vymyslet novou karetní hru, která bude využívat karty naprosto novým způsobem. Vzhledem k neznalosti těchto nových technik je nemůžeme zakomponovat do našeho řešení. Z těchto důvodů si napíšeme seznam konstrukcí, které náše řešení bude podporovat:

- libovolný počet hráčů - hry můžou být pro jednoho i více hráčů
- pouze tahové hry
- vytvoření vlastního balíčku karet
- nastavení umístění karet na hrací ploše - některé hry potřebují pouze karty na ruce a jiné zase potřebují karty vykládat na stůl

Zadávání pravidel by mělo být co nejflexibilnější, aby uživatel mohl vytvářet velkou škálu her. Na druhou stranu tato pravidla nesmí být příliš složitá, aby se uživatelům vyplatilo náš program používat místo naprogramování celé hry od začátku. Tyto vytvořené hry půjdou hrát po síti, nebo lze nahradit člověka umělou inteligencí. Na závěr od našeho řešení chceme, aby bylo platformě nezávislé kvůli co největší univerzálnosti.

1.2 Nové řešení

V této části si načrtneme řešení, které bude splňovat námi zvolené podmínky.

1.2.1 Platforma a programovací jazyk

Nejdříve se podíváme na platformní nezávislost. Jedná se určitě o rozumnou podmínku, protože k čemu by byl nástroj na tvorbu obecných karetních her, kdyby šel spustit jen na omezeném počtu počítačů? Z toho důvodů se jeví jako rozumná volba programovacího jazyka Java. Java je dnes velice rozšířená a není ani problém ji doinstalovat.

1.2.2 Grafika

Aby bylo možné karetní hry hrát, tak náš program musí obsahovat GUI. Můžeme udělat jen jednoduché GUI, které nám pouze umožní vytvářet a hrát hry. Taky se můžeme pokusit o složitější variantu, která bude zahrnovat některé grafické efekty. Složité GUI, ale není cílem naší práce. Navíc by k němu mohli být potřebné

externí knihovny. My se proto spokojíme pouze s použitím základní varianty. K té nám totiž stačí využít zabudovanou knihovnu Javy Swing[2]. Tuto knihovnu využijeme pro vytvoření návrhové části programu i herní části.

1.2.3 Síť

Síťová komunikace je nedílnou součástí našeho problému. Bylo by sice možné vytvořené hry hrát pouze proti UI, ale tím bychom hráče připravili o mnoho možností. Pro komunikaci po síti využijeme opět zabudovanou knihovnu Javy a to Net[3].

Abychom se mohli věnovat našim hrám a nemuseli se příliš starat o síťovou komunikaci, tak využijeme protokol TCP. Tento protokol nám zajistí spojitost a spolehlivost naší komunikace, o kterou bychom se museli jinak starat zvlášť. Podpora tohoto protokolu je již zabudována v námi zvolené knihovně.

1.2.4 Vzhled hry

Dále musíme vhodným způsobem navrhnout jak vytvořit vzhled hrací plochy. První možností je, že si uživatel nakliká komponenty zobrazované ve hře přímo na hrací plochu. Výhodou tohoto řešení je jednoduchost pro uživatele. Nevýhodou je, že by uživatel musel nastavit vzhled pro všechny možné počty hráčů zvlášť. Druhé řešení je, aby uživatel nastavil, kde se bude co vyskytovat. Toto řešení je pro uživatele trochu komplikovanější, protože nevidí ihned výsledek. Na druhou stranu stačí například nastavit, že každý hráč bude mít jen karty na ruce a už se nemusí starat o to, kolik vlastně hráčů bude ve skutečnosti hrát. Druhá varianta je i programově jednodušeji zpracovatelná, proto jsme se rozhodli zvolit ji.

1.2.5 Pravidla

Pravidla si musíme správně navrhnout, aby podle nich bylo možné řídit umělou inteligenci. První možností, co každého napadne, je naklikat si pravidla. Tato varianta je pro uživatele určitě vhodná z hlediska jednoduchosti. Problém je, že bychom mohli používat pouze předdefinované funkce. Pokud bychom v těchto funkcích nenašli to, co potřebujeme, tak bychom měli smůlu. Další možností je zadávat pravidla psanou formou. Nevýhodou tohoto postupu je, že se uživatel bude muset naučit zvláštní syntax pro zadávání pravidel. Druhou velkou nevýhodou je, že pro tuto variantu je potřeba základní znalost programování. Ovšem největší výhodou tohoto řešení je ohromná univerzálnost. Pravidla bychom sice chtěli udělat co nejjednodušší, ale není příliš vhodné to dělat na úkor univerzálnosti. Proto zvolíme tuto druhou variantu.

1.3 Již existující řešení

Většina programů je připravena pro konkrétní hru a neumožňuje modifikaci těchto her. Jediné, co takové programy umožňují, je přepínat mezi některými variantami. Vzhledem k tomu, že jsou tyto programy uzpůsobeny konkrétní hře, tak mohou mít lépe udělanou grafiku nebo umělou inteligenci, než obecná hra. Na druhou

stranu je univerzálnost asi naše nejdůležitější podmínka, tedy tyto programy nám příliš nevyhovují.

Další možností jsou programy bez kontroly pravidel. Takové programy nechávají veškeré hlídání pravidel na hrajících. Příkladem takového programu je Virtual Deck[1]. Tento typ programů plně splňuje modifikovatelnost her a zároveň splňuje i většinu našich ostatních podmínek. Ovšem nastává problém s umělou inteligencí. Vzhledem k tomu, že o hře nemáme vůbec žádnou informaci, tak nemůžeme vytvořit vhodný algoritmus. Mohli bychom nechat umělou inteligenci učit se podle tahů lidských hráčů. Následně by se umělá inteligence pokusila tyto tahy napodobit. Bohužel i u obyčejného Prší, které se hraje s 32 kartami, je příliš možností, jak karty rozdat. Tedy takováto umělá inteligence by nejspíš stejně podle pravidel nehrála.

2. Struktura programu

V této kapitole si rozebereme, jak bude celý program vypadat.

2.1 Části programu

2.1.1 Menu

Menu má dvě hlavní složky a to „Soubor“ a „Návrh“. Část „Návrh“ slouží k přepínaní mezi jednotlivými obrazovkami pro návrh nové hry resp. úpravy již existující hry. Část „Soubor“ slouží k přepnutí do obrazovky pro zahájení hry a k ukládání a načítaní vytvořených her.

2.1.2 Návrh

V sekci návrhu musíme umět vytvořit balíček karet, navrhnout vzhled hry a napsat pravidla. Volitelnou možností je napsání vlastní UI. To nám zobrazuje obrázek 2.1. Navržená hra se poté uloží do souboru, který se předává herní části.

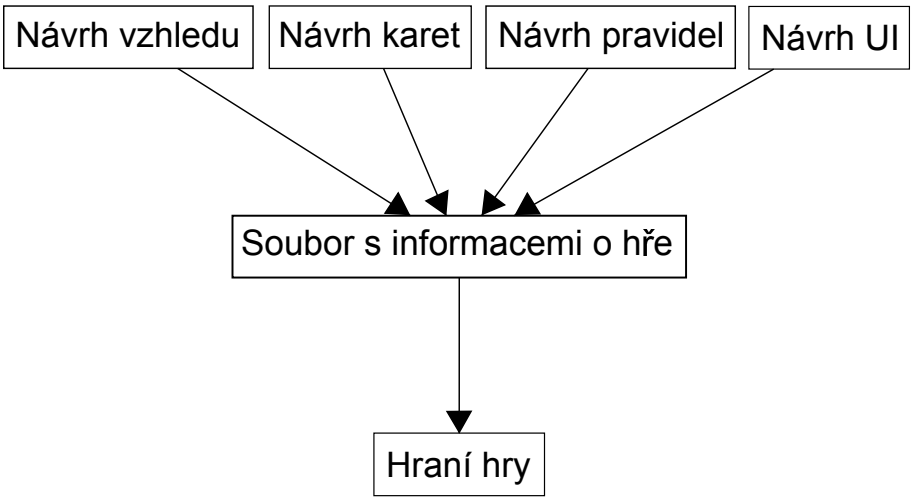
Každá karta se bude skládat ze dvou částí. První částí je obrázek, který uživatel uvidí při hře. Druhou částí jsou parametry karty. Protože v některých hrách se může jedna karta vyskytovat v balíčku více než jednou, tak každé kartě přiřadíme povinný parametr určující, kolikrát se bude v balíčku vyskytovat. Další parametry budou sloužit jako identifikátory dané karty. Jedná se například o barvu a hodnotu karty. Parametry si ale můžeme zvolit naprosto libovolně. V moderních karetních hrách se často používá více než jeden balíček karet, proto jeden parametr si můžeme zvolit jako identifikaci, ke kterému balíčku tato karta patří. V mnoha karetních hrách se používají stále stejné karty, proto také umožníme ukládání a načítání celých balíčků.

Nějakým vhodným způsobem musíme rozdělit herní plochu, abychom mohli jednoduše popsat, kde se co nachází. Základní dělení je na část středu a hráče. Středová část se bude nacházet uprostřed herní plochy a bude společná pro všechny hráče. V této oblasti se budou nacházet věci jako balíček na lízání karet nebo odhazovací balíček. Část hráče bude před každým hráčem. V ní se mohou nacházet karty na ruce nebo karty vyložené před hráčem. V těchto jednotlivých částech potom musíme nadefinovat jednotlivé oblasti, které nás budou při hře zajímat.

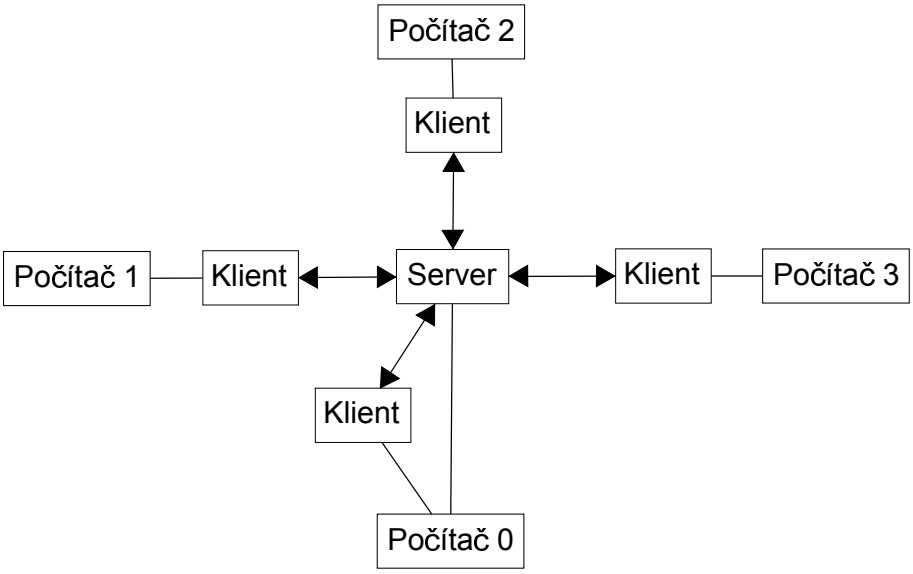
Pravidla a UI jsme se rozhodli zapisovat stejným způsobem a to v textové formě. I když nám to umožňuje dosáhnout vysoké univerzálnosti, tak je velice jednoduché udělat nějakou chybu. Proto si zde zavedeme kontrolu syntax.

2.1.3 Hra

Herní sekce bude sloužit k hraní již vytvořených her. Hru, kterou budeme chtít hrát, si budeme muset nejdříve vybrat ze seznamu vytvořených her a potom nastavit parametry. Parametry, které se musí nastavit, budou počet hráčů a jakou zvolit UI. Na závěr bude možné hru vytvořit a spustit tak server nebo se k již



Obrázek 2.1: Tvorba nové hry.



Obrázek 2.2: Síťová komunikace. Šipky značí komunikaci a jednoduché čáry značí příslušnost k danému počítači.

existující hře připojit a být pouze v roli klienta. Pokud se budeme chtít připojit, tak musíme zadat IP adresu počítače, ke kterému se připojujeme.

2.2 Síťová komunikace

Struktura síťové komunikace je zobrazena na obrázku 2.2.

Při zahájení hry se vytvoří vždy server. K němu se vždy připojují buď jednotliví hráči nebo umělá inteligence. Síťová komunikace se provádí i v případě, že hraje pouze jediný hráč. Velkou výhodou je, že server může být celkem jednoduchý a nerozlišovat mezi tím, jestli je k němu připojen živý hráč nebo UI. Ani nemusí rozlišovat, o kterou UI se jedná.

Aby se klienti mohli věnovat čistě hře a nemuseli se vůbec starat o pravidla (s výjimkou univerzální UI), tak veškerá kontrola pravidel se provádí na straně

serveru. Klient pouze serveru oznámí akci, kterou chce provést, a server, pokud to bude možné danou akci provede a informuje o tom všechny klienty. Výhodou tohoto způsobu je, že se jednoduše ohlídá, aby hrál pouze hráč, který je na řadě. Pokud totiž serveru přijde požadavek na provedení akce od hráče, který není na řadě, tak se jím vůbec nebude zabývat.

2.3 Umělá inteligence

Umělé inteligence budeme mít dva typy. Bude se jednat o univerzální a vlastní. Vlastní UI je velice jednoduchá. Jedná se o to, aby v době kdy se dostane daný hráč na řadu, aby umělá inteligence provedla zadaný kód. Přestože pravidla iUI se obojí zapisuje v textové formě, tak oboje bude mít jinou syntax, kterou se uživatel musí naučit. Z toho důvodu je vhodné syntax pravidel i UI navrhnout co nejpodobnější, aby uživatel neměl zase takové problémy.

Univerzální UI je o něco složitější. U ní není napevno dán nějaký jednoduchý kód, který by se provedl. Musíme zvolit vhodný algoritmus, který dokaže ze zadaných pravidel rozpoznat, jaké tahy jsou přípustné. Nakonec tento algoritmus by měl zvolit tah, který je pro daného hráče nejvýhodnější.

Umělá inteligence funguje tak, že si vytvoří vlastního klienta, který následně komunikuje se serverem. To bohužel vytváří problém přístupu k některým datům serveru. Některá data, jako například globální proměnné, chceme, aby byla přístupná. Z toho důvodu musí být u některých datových položek nastavena možnost veřejného přístupu k nim. Při vytváření vlastní umělé inteligence si tedy uživatel musí dávat pozor, aby opravdu přistupoval k položkám, které může daný hráč znát.

2.4 Ukládání

Vytvořené hry musí jít ukládat na disk. Takto uložené hry potom půjde hrát. Na ukládání her máme několik požadavků. První požadavek je, aby hry byly uložené v takovém formátu, aby z něj šly zjistit všechny potřebné informace o hře a ta šla potom hrát. Již vytvořenou hru chceme také umět načíst do programu, abychom ji mohli popřípadně upravit.

Všechny potřebné informace o hře se zadávají na obrazovkách pro návrh hry. Nejjednodušší by tedy bylo ukládat celé tyto obrazovky jako objekty na disk. Na to lze použít serializaci. Nevýhodou takovéhoto uložení je, že hry nemusí být funkční na jiném počítači. Přesto ukládání celých objektů není úplně špatná myšlenka, protože to umožňuje jednoduše načíst návrh hry naprosto ve stejném stavu, jako když byla uložena. Možnost jak uložit celé objekty, které by byly na jiném počítači obnovitelné, je ukládání ve formátu XML.

3. Algoritmy

V této kapitole si popíšeme hlavní algoritmy použité v programu.

3.1 Prohledávání stromu hry

Pro umělou inteligenci jsme zvolili metodu prohledávání stromu hry. Tato metoda je pro nás vhodná hlavně z toho důvodu, že ji lze použít bez příliš velkých znalostí o hře. Další důvod je, že pravidla můžeme zapisovat relativně jednoduše a přitom hodně univerzálně.

3.1.1 Co je strom hry?

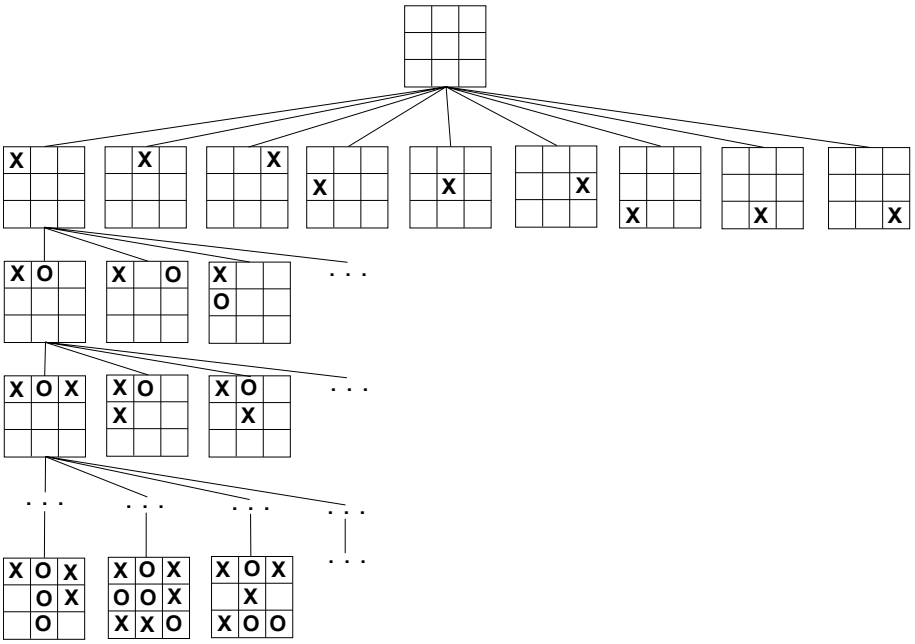
Nejdřív bychom se měli podívat co vlastně strom hry je. Strom hry si můžeme reprezentovat jako graf. V tomto grafu jsou vrcholy **stavy**, ve kterých se hra může nacházet, a hrany jsou **akce**, které je možné z daného stavu provést. **Počáteční stav** je stav, ve kterém se hra právě nachází, a v grafu je reprezentován kořenem. **Cílové stavy** jsou takové stavy, u kterých je splněna **cílová podmínka**, a v grafu jsou reprezentovány listy. Cílová podmínka je splněna vždy, když dojde ke konci hry. Počáteční stav spolu se všemi stavy, do kterých se z něj lze dostat nazveme **stavový prostor**. Potom **cesta** ve stavovém prostoru je cesta v grafu. Příkladem stromu hry (stavového prostoru) je strom na obrázku 3.1.

3.1.2 Minimax

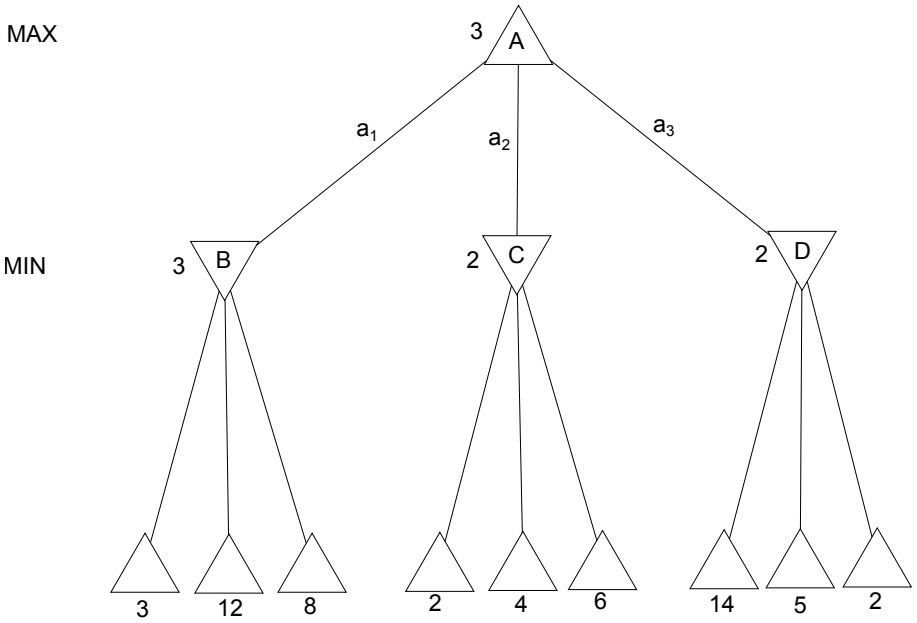
Základní verze minimaxu je pro dva hráče, které si nyní označíme jako MIN a MAX. Koncovým stavům lze pomocí **užitkové funkce** přiřadit numerickou hodnotu, která nám pro daného hráče říká, jak dopadl. Podmínkou pro tuto funkci je, aby v každém cílovém stavu byl součet hodnot ode všech hráčů konstantní. Například u šachů by tato funkce měla hodnoty 1 pro výhru , 0 pro prohru a 1/2 pro remízu. Součet takové funkce by byl potom vždy 1 (0+1, 1+0, 1/2+1/2).

Oba hráči se snaží najít optimální tah. Vzhledem k podmínce konstantního součtu nám stačí cílové stavy ohodnodit například jen pro MAX. Potom hráč MAX chce maximalizovat hodnotu užitkové funkce a hráč MIN minimalizovat. Nyní máme vhodně ohodnoceny cílové stavy, ale mi potřebujeme ohodnotit ještě všechny ostatní až po počáteční stav. Na to použijeme rekurzivní funkci MINIMAX(s), kde s je stav, kterému přiřazujeme hodnotu. Očividně tato funkce vrací hodnotu užitkové funkce pro cílové stavy, dále minimum z hodnot synů, pokud stav s odehrává hráč MIN, a na závěr vracíme maximum z hodnot synů, pokud stav s odpovídá hráči MAX. Pro ukázku se můžeme podívat na obrázek 3.2.

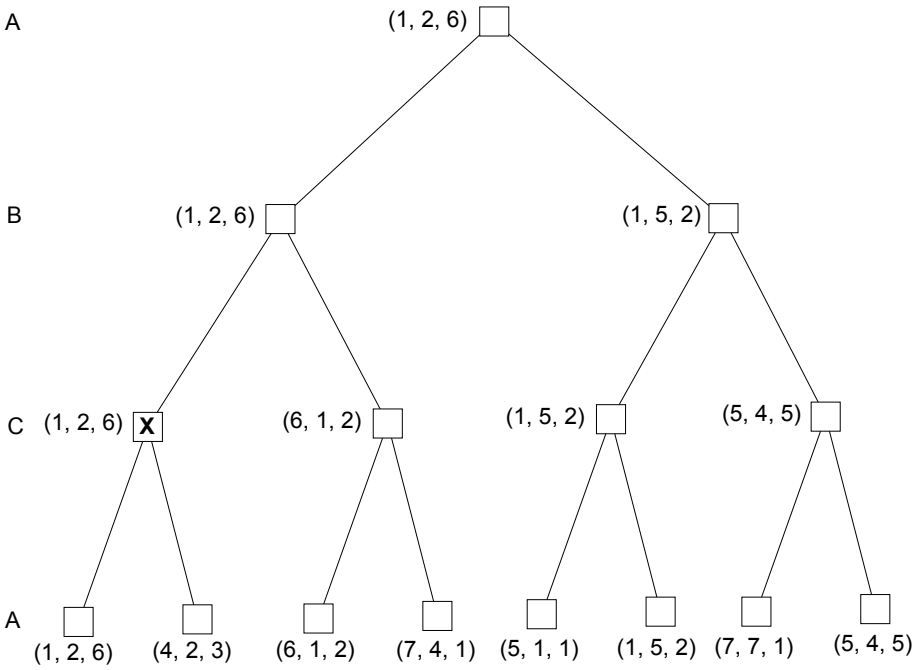
Když máme takto ohodnocen celý strom, potom zbývá, aby si hráč, který je na tahu vybral akci, která je pro něj optimální. Na obrázku 3.2, by to byla akce a_1.



Obrázek 3.1: Na obrázku je část stromu hry pro piškvorky na ploše 3x3. Počáteční stav je prázdná herní plocha. V první úrovni jsouí všechny možné umístění křížku. V další úrovni následují všechny možné umístění kolečka. Takto se postupuje až do cílových stavů. Cílové stavy můžeme vidět v poslední úrovni. Jedná se o stavy, kdy vyhrál hráč s křížky nebo kolečky, popřípadně nastala remíza.



Obrázek 3.2: Na obrázku je ukázka algoritmu minimax. Cílové stavy dostaneme ohodnocené z užitkové fukce. Ve stavech B, C a D hraje hráč MIN, který chce dosáhnout mnimální hodnoty. V uzlu A potom hraje hráč MAX, který chce dosáhnout naopak maximální hodnoty. Funkce nám tedy tímto způsobem ohodnotí celý strom.



Obrázek 3.3: Na obrázku je strom hry pro tři hráče označené A, B a C. Každý stav je ohodnocen vektorem hodnot (A, B, C).

3.1.3 Úpravy minimaxu pro hru více hráčů

Základní verze minimaxu nám jednoduchým způsobem nalezne optimální akci, kterou provést. Problém je, že nemůžeme tuto metodu použít pokud máme hru více jak dvou hráčů. Úprava algoritmu, ale naštěstní není vůbec složitá. Jediné co musíme udělat je nahradit jednu hodnotu, kterou nám funkce vracejí, vektorem hodnot. Pro hru dvou hráčů s konstantním součtem hodnot jsme mohli tento vektor nahradit jedinou hodnotou, protože tyto hodnoty byly vždy opačné.

Toto rozšíření si vysvětlíme na příkladu, který nám ukazuje obrázek 3.3. Jedná se o hru pro tři hráče. U každého vrcholu můžeme vidět vektor hodnot. První hodnota odpovídá hráči A, druhá hráči B a třetí hráči C. Každý hráč si vybere vždy takovou akci, aby maximalizoval svou hodnotu. Pokud se podíváme na stav označený X, potom hráč C si vybírá mezi (1, 2, 6) a (4, 2, 3). 6 je větší než 3, proto si vybere akci, která vede ke stavu (1, 2, 6). Stejným zbůsobem se doplní i zbytek stromu.

3.1.4 Úpravy minimaxu pro karetní hry

Až doposud jsme pracovali pouze s úplnou informací a bez náhody. Bohužel v karetních hrách většinou úplnou informaci nemáme, protože karty jsou zamíchány a po rozdání jsou některé karty známy pouze určitým hráčům. Většina karetních her zamíchá karty pouze na začátku. Nám tedy stačí vzít pouze všechny možné rozmíchání karet, které jsou pro nás neznáme, a potom pro každou tuto variantu spočítat minimax jako pro hru s úplnou informací. Následně stačí vybrat akci, která vychází průměrně nejlépe přes všechny možná rozdání karet.

Problémem této metody je, že pro karetní hry je všech možných rozdání karet takové množství, že by nebylo možné v reálném čase spočítat výsledek. Už spo-

čítaní výsledku pro jediné rozhození karet je velice obtížný úkol. Proto zvolíme metodu Monte Carlo. Nespočítáme všechna možná rozdání karet, ale náhodně zvolíme jen N možných rozdání. Když bude N=100, tak nám tato metoda již dá slušné výsledky. Urychlení je značné, protože například pro bridž je počet možností rozdání všech neznámých karet 10 400 600.

3.1.5 Heuristiky minimaxu

První metoda, na kterou se podíváme, je alfa-beta ořezávání. Tato metoda nám dokáže spočítat minimaxovou hodnotu bez toho, aby jsme procházeli celý strom hry. Průběh algoritmu je znázorněn na obrázku 3.4.

Nechť neohododnocení potomci uzlu C mají hodnoty x a y. Potom hodnota kořenového uzlu je

MINIMAX(koren)=max(min(3, 12, 8), min(2, x, y), min(14, 5, 2))
=max(3, min(2, x, y), 2)
=max(3, z, 2) kde z=min(2, x, y)$\leq$2
=3

Další námi uvedená metoda již bohužel nemusí dávat stejný výsledek jako minimaxový algoritmus. Nejprve si musíme říct, co to je **ohodnocovací funkce**. Jedná se o funkci, která nám v libovolném stavu hry odhadne hodnotu, kterou by vrátil minimaxový algoritmus, pokud by prošel celý podstrom.

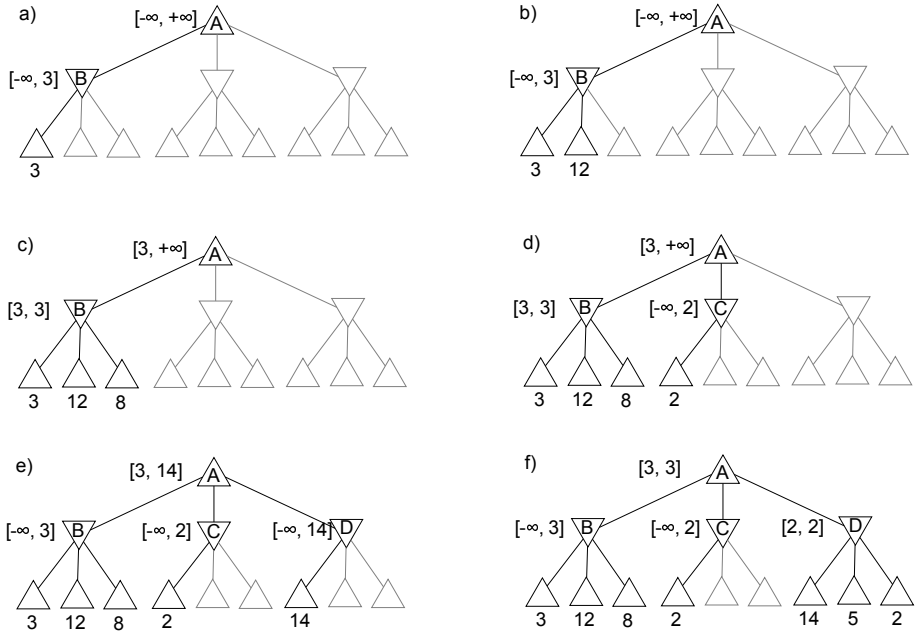
Při použití alfa-beta ořezávání sice nemusíme procházet celý strom, ale stále v některých větvích musíme dojít až k cílovému stavu. Proto se rozhodneme, že budeme strom prohledávat pouze do určité hloubky. Pokud narazíme na cílový stav před dosažením dané hloubky, tak vrátíme klasicky hodnotu užitkové funkce. Pokud ale dorazíme na námi zvolenou hloubku, aniž bychom narazili na cílový stav, potom vrátíme hodnotu ohodnocovácí funkce.

Velkou nevýhodou této metody je, že nám nedává přesný výsledek. Na druhou stranu nám velmi zrychlí celý algoritmus. V karetních hrách tuto metodu dokonce použít musíme, protože tam se nám může stát, že některé větve stromu budou nekonečné. Tedy pokud bychom se rozhodli nechat doběhnout minimax do konce, tak by nikdy neskončil.

3.1.6 Pravidla

Nyní, když máme popsán algoritmus pro umělou inteligenci, tak již můžeme vytvořit strukturu pravidel. Nejdříve si musíme rozmyslet, co všechno se v karetních hrách děje a jaké může být v počítači ovládání.

První akce co může hráč udělat je kliknout na nějakou kartu. Pokud hráč klikne na kartu na ruce, tak to může znamenat její zahrání, po kliknutí na balíček karet, to může znamenat líznutí karty, apod. V některých chvílích může být více možností co se může stát po kliknutí na kartu. Pro takové případy zavedeme tlačítka, která umožní přepínat mezi jednotlivými možnostmi. Například v Žolíkách může kliknutí na kartu znamenat odhození do odkládacího balíčku nebo vykládání na stůl. Další k čemu jsou tlačítka dobrá, je výběr hráčů v některých hrách. Pravidla si tedy zavedeme jako seznam akcí ,která se mají stát po určité akci hráče.



Obrázek 3.4: a) Prohledali jsme zatím pouze prvního následníka B, který má hodnotu 3. B patří hráči MIN, proto bude mít maximálně hodnotu 3. b) Druhý následník B má hodnotu 12. MIN zavrhne tuto cestu, tedy maximální hodnota bude stále 3. c) Poslední následník B má hodnotu 8. Viděli jsme již všechny následníky, proto uzel B hráče MIN bude mít právě hodnotu 3. d) První následník C má hodnotu 2. MIN zvolí tedy tak, že C bude mít maximálně hodnotu 2. Víme již, že pod uzlem B je hodnota 3 , což je více než 2. Tedy ostatní následníky pod C již nemusíme procházet. Toto byla ukázka alfa-beta ořezávání. e) První následník D má hodnotu 14, tedy D bude mít maximálně hodnotu 14. To je ale více než alternativa z uzlu B pro hráče MAX, tedy musíme pokračovat v prohledávání pod uzlem D. Protože už máme hranice pro všechny syny kořene, tak můžeme kořen lépe ohraničit. f) Druhý následník D je 5, tedy musíme stále pokračovat v hledání. Třetí následník D je 2, tedy i hodnota D bude přesně 2. MAX si potom tedy vybere akci k uzlu B s hodnotou 3.

První, co tedy každé pravidlo musí obsahovat, je místo, kde došlo ke kliknutí na kartu, resp. na tlačítko. U tlačítek nám to stačí, ale u karet toho potřebujeme víc. Další, co u každého pravidla musí být je podmínka. Akce uvedené u pravidla se provedou pouze pokud bude splněna daná podmínka. Pomocí podmínky můžeme rozlišovat jednotlivé karty. Někdy mohou být splněné dvě a více podmínek najednou. V takovém případě budeme chtít, aby se provedlo jen jedno pravidlo. Které ale vybrat? Nejlepší by asi bylo vybrat to, které bude mít nejspecifičtější podmínku. To ale nemusí být zrovna jednoduché určit, když o hře nemáme skoro žádné informace. Z těchto důvodů u každého pravidla bude uvedena priorita. Pokud by šlo provést více jak jedno pravidlo, tak se provede to s největší prioritou.

Nyní rozeznáváme již akce hráče, ale existují ještě dvě zvláštní pravidla, které musíme nastavit. První z nich je seznam akcí, co se má stát na začátku hry. Do tohoto pravidla přijdou věci jako zamíchání a rozdání karet. Druhé speciální pravidlo bude mít dvě části. Po každé akci, kterou hráč provede je nutno zkontrolovat, jestli nenastal konec hry, k čemuž bude sloužit první část tohoto pravidla. Druhá část tohoto pravidla se provede pouze pokud nastal konec hry. Ta slouží k tomu, aby se opět došlo k rozdání karet.

Nyní máme již připraven základ pravidel. Víme, co vše se má stát po jednotlivých akcích hráče, i kdy je konec hry. Ještě je potřeba určit, kdo vyhrál a popřípadně jak moc vyhrál. V některých hrách, jako například v Mariáši, totiž nezáleží jen na výhře a prohře, ale i na tom kolik hráč nasbíral bodů. Při psaní pravidel musí tedy uživatel ještě nastavit ohodnocovací funkci. Vzhledem k tomu, že se tato funkce bude používat pouze na závěr hry, tak její definování umístíme do první části koncového pravidla.

3.2 Zásobníkový automat

V této kapitole se podíváme na to, co je zásobníkový automat. Ten budeme používat ke kontrole syntaktické správnosti pravidel a parsování do připravených datových struktur.

Než si řekneme co je zásobníkový automat, tak si musíme zavést několik pojmů. **Terminál** je znak z cílové abecedy. **Neterminál** je znak z pomocné abecedy. Neterminály se mohou přepisovat na jiné znaky. **Gramatika** je systém přepisovacích pravidel. **Bezkontextová gramatika** je gramatika, která obsahuje pouze pravidla, která přepíší jeden neterminál na libovolný počet znaků, a to jak terminálů tak neterminálů.

Zásobníkový automat je mechanismus, který nám řekne, zda daný vstup lze vygenerovat zadanou bezkontextovou gramatikou. Zásobníkový automat se skládá ze tří částí. První je čtení vstupu, druhá je řídící jednotka a poslední je zásobník. Zásobníkový automat v každém kroku vyjme vrchní znak ze zásobníku, a potom se rozhodne co udělat. Může udělat několik věcí. První je, že pouze vloží nové znaky na zásobník (nemusí také vložit nic) a přesune se do určitého stavu. Další možností je přečtení znaku ze vstupu a podle něj se rozhodnout, jak upravit zásobník a aktuální stav automatu. Poslední možností je, že automat nemá žádnou možnost co udělat, a to znamená, že vstup nelze gramatikou vygenerovat.

Na tuto gramatiku máme požadavek,aby byla jednoznačná. Jednoznačná bezkontextová gramatika je taková, která dokáže každý možný vstup vygenerovat pouze jediným možným způsobem.

4. Programátorská dokumentace

4.1 Ukládání

Hry se budou ukládat ve formátu XML. Na to v Javě existují třídy XMLEncoder[5] a XMLDecoder[6]. Jedinou slabou stránkou těchto tříd je, že dokáží pracovat pouze s JavaBeans. Všechny objekty, které budeme chtít ukládat, tedy musí mít nulární konstruktor a všechny položky, které se budou ukládat musí být privátní a mít Getter a Setter.

Budeme ukládat celé objekty, kterými budou jednotlivé obrazovky pro návrh hry. Těchto obrazovek je, ale více a my bychom chtěli, aby hra byla pouze jediný soubor. Proto zavedeme třídu Informace package data. Tato třída slouží pouze uložení dat. Obsahuje v sobě všechny obrazovky pro návrh hry. Pokud budeme chtít někdy hru uložit nebo načíst, tak budeme ukládat resp. načítat objekt, který bude instancí této třídy.

4.2 Klient

Máme dva hlavní typy klientů. První je klient, kterého řídí UI, a druhého řídí hráč. V této kapitole se budeme zabývat druhým typem klienta. Celý je naimplementovaný ve třídě Klient package sit.

4.2.1 Vzhled

Toto je jediný typ klienta, který musí zobrazovat hru. Celá hra je zobrazena v jednom JDialogu, který se vytváří a zobrazuje pomocí metody createAndShowDialog(). Jednotlivé oblasti herní plochy jsou reprezentovány pomocí pomocné třídy Hrac. V dialogu využíváme Border layout. Uprostřed je vždy část, která je označena jako střed. Po obvodu jsou potom oblasti pro jednotlivé hráče. Jejich umístění záleží na celkovém počtu hrajících hráčů. Hráč, pro kterého se klient vytváří, je vždy umístěn ve spodní části dialogu. V případě, že hrají dva hráči, tak druhý je umístěn v horní části dialogu. Ve třech hráčích zbylí dva hráči jsou umístěni vlevo a vpravo. Při čtyřech hráčích jsou obsazeny všechny strany. Kvůli přehlednosti není povoleno, aby hrálo více jak čtyři hráči.

V informacích o hře máme sice obrázky jednotlivých karet, ale to nám nestačí. Některé karty se mohou ve hře vyskytovat vícekrát. Z toho důvodu máme seznam JLabelů, které obsahují obrázky jednotlivých karet. Tyto seznamy musíme mít dva. jeden je pro přední stranu karet a druhý pro zadní stranu. O vytvoření těchto seznamů se nám stará metoda pripravit(). Tyto seznamy musí být vždy a všude vytvořeny naprosto stejným způsobem, protože karty jsou později identifikovány pouze pomocí pořadí v seznamu.

4.2.2 Třída Hrac

Toto je pouze pomocná třída. Rozšiřuje JPanel. Instance této třídy se potom vkládají na obrazovku a reprezentují jednotlivé hráče. Metoda vloz() vloží do této třídy novou komponentu na zadané souřadnice a se zadaným jménem. Tato

komponenta je reprezentována jako JPanel s Flow layoutem. Do těchto Flow layoutů se potom vkládají jednotlivé karty, které jsou reprezentovány ikonami JLabelů.

Oblasti jsou u každého hráče stejně umístěny i pojmenovány, proto je zde vytvořena metoda copy(), která zkopíruje tuto třídu. Kopíruje se ale jen struktura oblastí, nikoliv jejich obsah.

4.2.3 Práce klienta

Klient běží ve zvláštním vlákně. V tomto vlákně nejdříve dojde k propojení klienta se Serverem. Následně se vytvoří dialog pro zobrazování hry. Nakonec se v nekonečném cyklu čte vstup od serveru.

Klient nedělá žádné výpočty týkající se průběhu hry. Pouze zobrazuje změny, o kterých mu řekl server, a posílá informace o kartách, které by chtěl hráč zahrát. Tyto změny se provádí v metodě proved().

Metoda proved

Server a klient si vyměňují veškeré informace v textovém formátu. Jednotlivé části každého příkazu jsou odděleny mezerou.

První požadavek, co může server poslat, je zobrazení nějaké karty, textu nebo tlačítka. V takovém případě pošle „ins“ a následuje číslo hráče a místo, kde má dojít k zobrazení. Následuje „lab“ resp. „but“, pokud se jedná o text resp. tlačítko. Po nich následuje text, který se má zobrazit. U tlačítka se musí nastavit ActionListener, který odešle serveru zprávu, když bude tlačítko stisknuté. Tato zpráva bude mít formát „button“ a potom následuje číslo hráče a název oblasti, kde došlo ke stisknutí tlačítka. Kromě textu a tlačítek se může ještě zobrazit karta. V takovém případě se za informací o místě nachází číslo karty, která se má zobrazit, a jestli se má zobrazit její přední nebo zadní strana. Takto zobrazené kartě se musí přiřadit listener, který odešle zprávu serveru, pokud hráč na danou kartu kliknul. Zpráva má formát „klik“ a následuje číslo hráče s místem, kde ke kliknutí došlo, a číslo karty.

Další možností, co server může chtít udělat, je odebrání karty, textu nebo tlačítka z nějakého místa. V takovém případě server pošle „del“ následované číslem hráče a místem, kde k odebrání má dojít. Následuje „lab“ resp. „but“, pokud se má odebrat text resp. tlačítko. Může také následovat číslo karty, která se má odebrat, s informací o tom jestli byla vidět její vrchní nebo spodní část.

Server může poslat „show“. V takovém případě se zobrazí dialog s tlačítky, kde každé tlačítko má text odpovídající dalším parametrům za „show“. Toto slouží k výběru z nějakých možností. Ke každému tlačítku je přidán listener, který odešle serveru informaci o tom jaké tlačítko bylo zmáčknuto. Tato informace je pouze text tlačítka, na které se kliklo.

Server také může poslat „showinf“. V takovém případě se zobrazí dialog, který bude zobrazovat text poslaný jako parametr za „showinf“. Tento dialog je čistě informační.

Poslední možností, co může klient od serveru obdržet, je „next“. To nám určuje, že na řadě je nějaký další hráč. Číslo hráče, o kterého se jedná, se dozvíme z parametru. Tato informace slouží k zobrazení aktuálního hrajícího hráče v titulku dialogu pro zobrazení hry. Je to z toho důvodu, aby hráč věděl, kdo je na řadě.

Obrázek 4.1: Menu hry.

4.3 GUI

V této kapitole si popíšeme GUI používané pro návrh her. Všechny třídy, které jsou k tomu potřeba se nacházejí v package GUI.

4.3.1 Třída Main

Třída Main obsahuje metodu main, která pouze volá metodu createAndShowGUI(). Metoda createAndShowGUI vytvoří a poté zobrazí celý program. Konkrétně vytvoří instanci třídy PanelHry, která obsahuje základní obrazovku. V této metodě se volá také metoda createMenu, která vytvoří menu, které je na obrázku 4.1. Část Návrh slouží pouze k přepínaní mezi jednotlivými částmi pro návrh hry. Část Soubor obsahuje možnosti pro vytvoření nové hry, uložení a načtení již existující hry, ukončení programu a přepnutí do obrazovky pro zahájení hraní námi vytvořené hry.

4.3.2 Třída PanelHry

Tato třída rozšiřuje JPanel a obsahuje obrazovku pro zahájení hry. Náčrt vzhledu této obrazovky je na obrázku 4.2. První JComboBox slouži k výběru hry, kterou chce hráč hrát. Následují JComboBoxy pro výběr počtu hráčů a počtu hráčů, které bude ovládat počítač. Následuje JCheckBox, který je zaškrtlý, pokud chce hráč používat umělou inteligenci speciálně napsanou pro danou hru. Dále následuje JLabel s IP adresou počítače. Pokud se chce hráč připojit k již existující hře, tak vyplní JTextField IP adresou počítače, na kterém běží Server. Následují tlačítka pro vytvoření hry a připojení k již existující hře.

4.3.3 Třída PanelKaret

Tato třída rozšiřuje JPanel a obsahuje obrazovku pro tvorbu balíčku karet. Vzhled obrazovky je načrtnut na obrázku 4.3. V horní části jsou tlačítka pro uložení balíčku pro budoucí použití a načtení balíčku pro načtení již existujícího balíčku. Další tlačítko slouží ke změně parametrů karet. Každá karta má parametr název,

Obrázek 4.2: Obrazovka pro zahájení vytvořené hry.

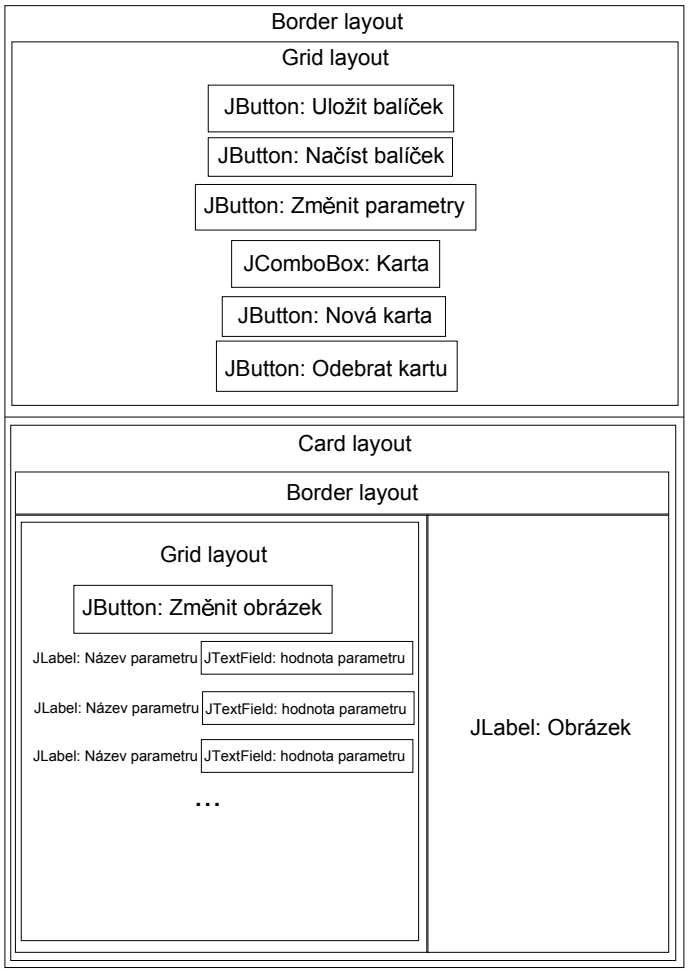
který je jednoznačný identifikátor karty, a počet, který určuje kolikrát se karta v balíčku vyskytuje. Další parametry jsou libovolné informace, které chceme u každé karty mít. Změna parametrů se provádí ve zvláštním dialogu, který se vytvoří a spustí metodou createAndShowDialog(). Na obrazovce následuje JComboBox, kterým se přepíná mezi jednotlivými kartami v balíčku. Poslední dvě tlačítka v horní části obrazovky jsou pro přidání resp. odebrání karty z balíčku. Pro přidání karty do balíčku máme metodu vloz().

Spodní část obrazovky obsahuje CardLayout, kde v každé záložce je JPanel s informacemi o dané kartě. Tento JPanel je rozdělen na dvě části. V levé se nachází tlačítko pro změnu obrázku karty. Tyto obrázky musí být uloženy v adresáři „obrazky“. Levá část dále obsahuje seznam parametrů se svými hodnotami. U JTextFieldu pro zadávání názvu karty je přidán ActionListener, který změní název v JComboBoxu pro výběr karty. V pravé části potom už je jen JLabel, který slouží k zobrazení obrázku karty.

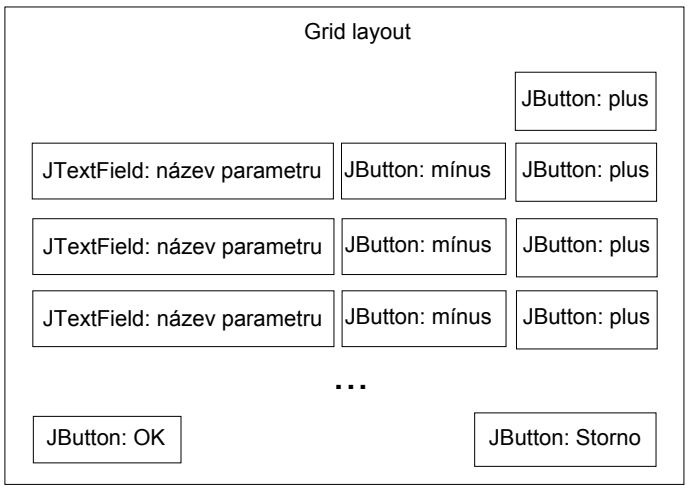
V této části se ukazuje jeden problém, kterým je ukládání her do XML. Všechna data můžeme do XML uložit, ale obrázky ne. Každý obrázek se nám uloží pouze jako absolutní cesta k obrázku na disku. Z toho důvodu musí být všechny obrázky ve stejném adresáři, ke kterému budeme znát relativní cestu. V návrhu zobrazujeme obrázky karet jako ikony JLabelů. Jako jednotlivé JLabely je i ukládáme. Jako jméno jednotlivých JLabelů nastavíme název obrázku, pod kterým je uložen na disku. Nyní již můžeme na libovolném počítači načíst obrázky zpátky do programu.

Metoda createAndShowDialog

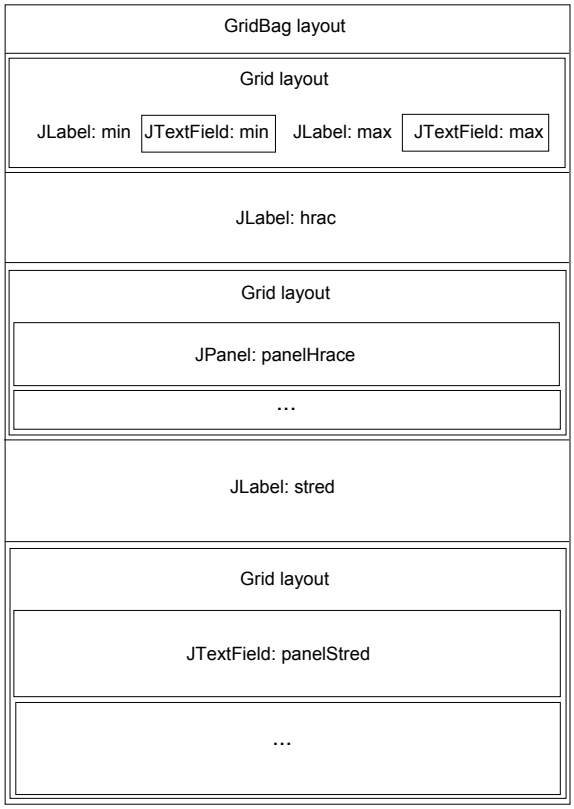
Tato metoda vytvoří a zobrazí dialog, který je načrtnut na obrázku 4.4. Pokud uživatel doposud nezadal žádné parametry, tak celý dialog bude obsahovat pouze tlačítka plus, OK a Storno. Pokud uživatel již edituje dříve zadané parametry, pak se mu zobrazí příslušný počet řádků. V každém řádku je JTextField pro zadání názvu parametru a tlačítka plus a mínus pro přidání resp. odebrání parametru. Po kliknutí na tlačítko OK se aplikují změny do návrhu karet a po kliknutí na Storno se všechny změny zahodí.



Obrázek 4.3: Vzhled obrazovky pro návrh balíčku karet.



Obrázek 4.4: Vzhled dialogu pro návrh parametrů.



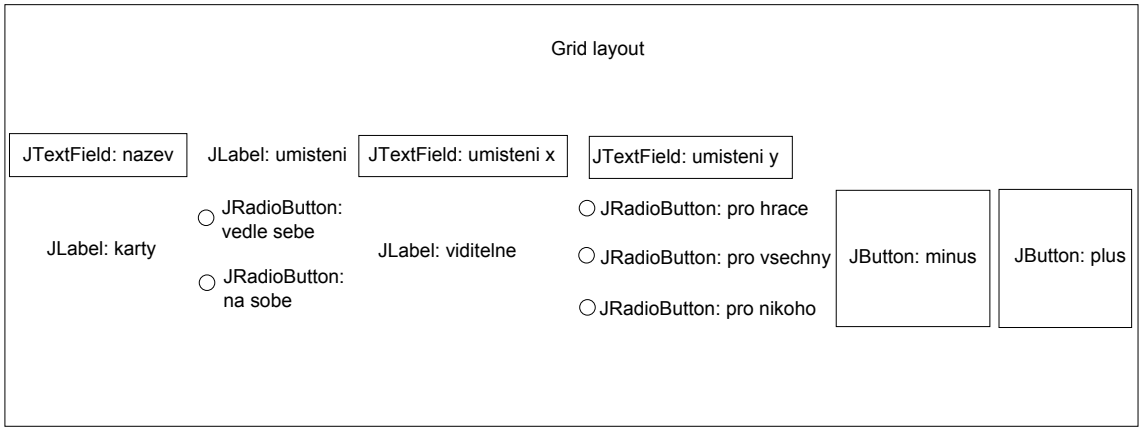
Obrázek 4.5: Obrazovka pro návrh vzhledu hry.

4.3.4 Třída PanelVzhledu

Tato třída rozšiřuje JPanel a obsahuje obrazovku pro návrh vzhledu hry. Obrazovka je načrtnuta na obrázku 4.5. Tato obrazovka je rozdělena na tři hlavní části. Horní část slouží k zápisu minimálního a maximálního počtu hráčů, které chceme pro hru povolit. Další dvě části slouží k určení toho, co bude před každým hráčem a co mezi nimi. Prostřední část má nejdříve nadpis, který nám říká, že se jedná o návrh u hráčů. Následuje Grid layout, ve kterém se za sebou nachází JPanely s informacemi o jednotlivých oblastech. Tyto JPanely nám vytváří metoda panelHrace(). Poslední část vypadá úplně stejně jako prostřední, jen s tím rozdílem, že nadpis nám určuje, že se jedná o návrh středu a jednotlivé JPanely se vytváří pomocí metody panelStred().

Metoda panelHrace

Vzhled tohoto panelu je nakreslen na obrázku 4.6. Každý jednotlivý JPanel popisuje jednu konkrétní oblast, která se nachází před každým hráčem. Nejdříve se do JTextFieldů vyplní název oblasti a souřadnice, kde bude oblast umístěna. V druhém řádku jsou JRadioButtony, které slouží k přepínání mezi jednotlivými možnostmi viditelnosti karet. Následují tlačítka plus a mínus pro přidání nového resp. odebrání stávajícího JPanelu.



Obrázek 4.6: JPanel pro návrh oblasti před hráčem.

Metoda panelStred

Tato metoda je skoro stejná jako metoda panelHrace(). Hlavní rozdíl je, že slouží k návrhu oblastí, které jsou uprostřed herní plochy a nepatří žádnému hráči. Výsledný JPanel tedy vypadá jako na obrázku 4.6. S tím rozdílem, že neobsahuje JRadioButton pro viditelnost pro hráče.

4.3.5 Třída PanelPravidel a PanelUI

Tyto dvě třídy jsou rozšířením JPanelu a reprezentují obrazovky pro návrh pravidel a vlastní UI. Obě obrazovky mají stejný vzhled jak je nakresleno na obrázku 4.7. Do horní JTextArea se píší pravidla resp. UI. Po kliknutí na tlačítko kontrola se spustí metoda zkontroluj().

Metoda zkontroluj

Tato metoda využívá třídy Automat balíčku inteligence ke zkontrolování správné syntax pravidel resp.UI. První nalezenou chybu následně vypíše do spodní JTextArea v obrazovce pro tvorbu pravidel resp. UI.

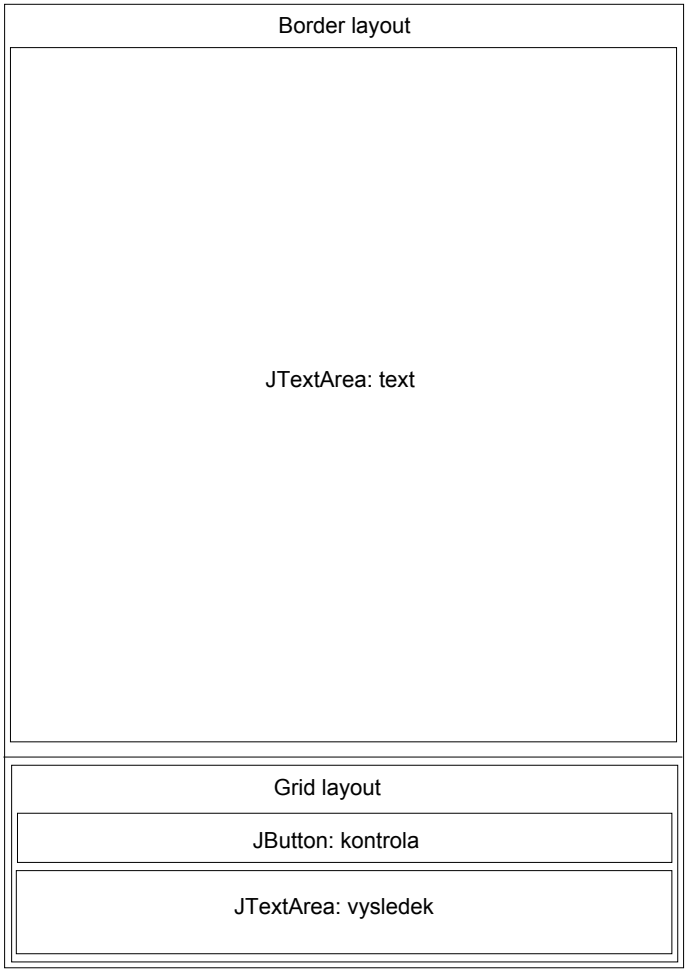
4.4 Prekladac

V této kapitole se podíváme jak zkontrolovat syntaktickou správnost pravidel a jak je naparsovat do připravených datových struktur. Použijeme na to upravený zásobníkový automat.

Všechny části co se týkají kontroly a parsování vstupu jsou obsaženy ve třídě Automat package inteligence.

4.4.1 Datové struktury

Pro správné vyhodnocování pravidel potřebujeme znát hlavičky pravidel, obsah pravidel a jaké používají proměnné. Proměnné se ukládají jako HashMapy, kde klíčem je název proměnné a obsahem je hodnota dané proměnné. Na stejné úrovni



Obrázek 4.7: Obrazovka pro návrh pravidel a UI.

jako globální proměnné ještě zjišťujeme, jak lze porovnávat Stringy. Stringy, které chceme porovnávat, se ukládají do seznamu a to od nejnižší hodnoty po nejvyšší.

Pravidla jsou uložena v trojrozměrném poli Stringů. Každý příkaz lze rozložit na několik Stringů, kde první String je informace o tom, o jaký se jedná příkaz, a další Stringy jsou parametry příkazu. Tedy pro každý příkaz potřebujeme jednorozměrné pole. Příkazy jdou za sebou v seznamu podle toho jak se mají vykonávat. To už máme tedy dvojrozměrné pole. Třetí rozměr je potřeba k tomu, abychom rozlišili jednotlivá pravidla resp. části pravidel.

Poslední částí jsou hlavičky jednotlivých pravidel. Ty jsou uloženy v HashMapě, kde klíčem je pole Stringů s jednotlivými parametry hlavičky a hodnotou je index do seznamu pravidel. Tento klíč musí kromě parametrů hlavičky obsahovat ještě několik věcí. První musí obsahovat o jaký typ hlavičky se jedná. Další věc, co musí obsahovat, je seznam lokálních proměnných, které se u daného pravidla deklarují.

V seznamu pravidel se nemusí nacházet jen celá pravidla, ale třeba i jen jejich části. Například obsah cyklu se uloží do seznamu jako zvláštní pravidlo a k příkazu se přidá parametr s indexem na toto pravidlo.

4.4.2 Gramatika

Třída musí obsahovat dvě různé gramatiky. První pro pravidla a druhou pro UI. Gramatika je zapsána v HashMapě. Klíčem této mapy je neterminál, pro který hledáme změnu, obsahem je dvourozměrné pole Stringů. Jedná se o seznam seznamů na co se neterminál mění. Zde dochází k drobné změně oproti normálnímu zásobníkovému automatu a automatům obecně. Neterminál se nemusí změnit jen na neterminály a terminály, ale i na jiné pomocné znaky.

Každý neterminál začíná písmenem „n“ a každy terminál písmenem „t“. Pomocným znakem je „m“. Tento znak říká, že může následovat libovolně mezerem, tabulátorů a nových řádků. Dalšími pomocnými znaky jsou „sx“, kde „x“ značí číslo od 0 do 8. s0 značí Stringy pro porovnávání. s1 až s4 značí, že následují deklarace proměnných a to lokálních intových resp. Stringových resp. globálních intových resp. Stringových. s5 značí, že následuje String složený z písmen anglické abecedy, čísel a podtržítka. s6 značí, že následuje libovolné číslo. Pokud nějaký typ proměnných není zadeklarovaný, potom s7 vloží na jeho místo prázdný seznam. Nakonec s8 vynuluje lokální proměnné.

Při vytváření gramatiky pro zadávání vlastní UI, se musí spustit i kontrola pravidel. To je nutné, abychom věděli jaké byly použity globální proměnné. UI sice tyto globální proměnné nemůže měnit, ale někdy může potřebovat se podle nich rozhodovat. Problémem tohoto přístupu je, že na některých systémech může docházet k problémům s pamětí při kontrole syntax UI.

4.4.3 Překlad

Výpočet se zahajuje metodou vypocet(). Tato metoda slouží k inicializaci výpočtu a pokud je v zadaném vstupu chyba, tak vrátí informaci o tom, kde k chybě došlo a o jakou chybu šlo.

Hlavní výpočet se děje v rekurzivní metodě zpracuj(). Tato metoda na začátku dostane celá pravidla resp. UI. Podobně jako zásobníkový automat v každém

kroku odebere jeden symbol ze zásobníku. Následně, pokud se jedná o terminál, tak ho porovná se začátkem vstupu, a pokud souhlasí, tak ze začátku vstupu odebere vše, co odpovídá tomuto terminálu a dojde k zanoření do rekurze. Pokud je na zásobníku neterminál, potom se na zásobník přidají symboly, na které se může neterminál změnit. Pokud se stane, že terminál nebude odpovídat začátku vstupu, tak dojde k vynoření z rekurze, a zkusí se předchozí neterminály změnit na jiné symboly. Pokud dojde k vynoření až k terminálu, který uspěl, tak se vrátí chyba. V případě, že na zásobníku je jiný symbol než terminál a neterminál, tak se provede to čemu odpovídá.

4.5 Server

Na serveru se provádí veškeré výpočty týkající se kontroly pravidel. Server musí umět komunikovat s více než jedním klientem, proto je server vícevláknový. Všechny části týkající se serveru jsou ve třídě Server package sit. Kvůli tomu, že se jedná o vícevláknový server, tak tato třída dědí od třídy Thread.

4.5.1 Vytvoření

Při vytváření serveru nejdříve dojde ke zkontrolování syntaktické správnosti pravidel a vlastní UI. Následně se pomocí metody pripravit() naplní připravené datové struktury. Následně server čeká na připojení jednotlivých klientů a pro každého vytvoří instanci třídy Server, která s ním bude komunikovat. Pokud hráč nastavil, že chce hrát s UI, tak server vytvoří instance těchto UI, které budou následně komunikovat s vytvořenou instancí Serveru. Vzhledem k tomu, že zahájení hry inicializoval hráč, který bude chtít hrát, tak se vytváří ještě instance Klienta pro tohoto hráče.

Jakmile se připojí všichni hráči, tak se spustí všechna vlákna serveru a provedou se příkazy zadané pod pravidlem „start“.

4.5.2 Práce serveru

Server pouze v cyklu čeká, až mu klient pošle požadavek na nějakou akci. Pokud pošle požadavek klient, který zrovna není na řadě, tak se tento požadavek vůbec nevyhodnocuje a čeká se rovnou na další. Tím je zabezpečeno, že hraje opravdu jen hráč, který je na řadě, a jednotlivá vlákna serveru si nemohou vzájemně pod rukou měnit data.

Server potom následně vybere ze všech pravidel to, které nejlépe odpovídá příchozímu požadavku. Přitom se připraví pro toto pravidlo datové struktury pro lokální proměnné. Nakonec se zavolá metoda proved(), která provede příkazy obsažené v pravidle. Pokud není nalezeno pravidlo, které by odpovídalo příchozímu požadavku, tak se nic nestane a čeká se na další požadavek.

4.6 Vlastní UI

Vše co se týka fungování vlastní umělé lze nalézt ve třídě UINapsana package inteligence. Tato umělá inteligence vždy, když se dostane na řadu, tak provede

zadané příkazy.

V cyklu tedy čte vstup od serveru. Pokud přijde informace, že daný hráč je na řadě, tak se zavolá metoda proved(), která provede zadané příkazy podobně jako Server. Rozdíl oproti Serveru je, že se nemusí vybírat vhodné pravidlo, protože je zde pouze jediné.

Problémem této umělé inteligence je, že výpočet je až příliš rychlý. Když by totiž UI odehrávala více tahů za sebou, tak by lidský hráč neměl šanci postřehnout, jaké karty vlastně UI zahrála. Z toho důvodu se uspí vlákno s výpočtem ještě před tím, než k výpočtu dojde a to na dobu 500 ms.

4.7 Univerzální UI

Velkou částí této prací je vytvoření UI, která bude schopná hrát libovolnou hru. Tato UI je naimplementovaná ve třídě UIUniverzalni package inteligence.

4.7.1 Pomocné třídy

UI využívá dvě pomocné třídy a obě jsou implementovány jako vnitřní třídy. První třídou je třída Karta, jejíž instance reprezentuje jednotlivé karty herního balíčku. Kromě parametrů karty ještě třída obsahuje informaci o tom, zda UI ví kde se tato karta momentálně nachází. Na začátku UI neví o umístění žádné z karet. Později se dozvídá o kartách které jsou pro danou UI momentálně viditelné. Avšak aby UI lépe vyhodnocovala situaci, tak si musí pamatovat i karty, které už nevidí. Na druhou stranu v některých hrách může dojít k zamíchání některých karet zpátky do balíčku, a potom by si UI měla přestat pamatovat, kde se daná karta nachází. Server informuje o kartách, které má UI zapomenout zprávou ve formátu „shuffle x“, kde x je číslo karty, kterou má zapomenout.

Druhou pomocnou třídou je třída Stav. V této třídě se pamatuje stav hry. Při vytváření stromu hry je v každém uzlu jeden Stav.

4.7.2 Průběh algoritmu

Jakmile přijde nějaká zpráva od serveru, tak se zpracuje pomocí metody aktualizuj(). Tato metoda aktualizuje aktuální Stav hry nebo pokud se UI dozví, že je na řadě, tak odehraje.

Metoda zacatek

O výběr nejvhodnějšího tahu se stará metoda zacatek(). Protože strom, který by se pro hru vytvářel, by mohl mít příliš velkou hloubku na to, aby výpočet proběhl v rozumném čase, tak tato metoda dostává jako parametr maximální hloubku stromu. Tento parametr je stanoven na 5. Při vyšších hodnotách by totiž byl výpočet již hodně pomalý.

UI „nezná“ umístění všech karet, proto se v metodě začátek náhodně vybere 100 rozložení neznámých karet. Vznikne nám tedy 100 různých stavů hry, pro které už můžeme spustit minimaxový algoritmus. Algoritmus na první úrovni se provádí ještě v metodě zacatek(). Pro ostatní úrovně stromu se již používá rekurzivní metoda minimax().

Metoda minimax

Vzhledem k tomu, že nám jde pouze o ohodnocení stromu, tak je zbytečné pamatovat si celý strom v paměti. Nám stačí pouze vracet hodnotu ohodnocovací funkce.

To, že algoritmus končí v hloubce určené parametrem, není úplně pravda. Parametr spíše určuje počet pravidel, která se provedou. To je vlastně stejné číslo jako počet tahů, ale hloubka stromu může být i vyšší. Uprostřed pravidla se může nacházet zamíchání, které zvýší hloubku o jedna, protože musíme opět zkusit možné varianty, jak byly karty zamíchány jako v úplně první úrovni stromu. Kromě této možnosti existuje ještě druhá situace, kdy může dojít k rozvětvení stromu uprostřed tahu. Tato situace nastane při použití příkazu „show“, který umožňuje vybrat nějakou hodnotu.

4.8 Spuštení

Vzhledem k tomu, že náš program je v zásadě hra, tak je nepraktické, aby byl spouštěn z příkazové řádky. Proto vytvoříme spustitelný jar soubor. Toto řešení má ale také své problémy. Kontrola syntaxe pravidel a jejich parsování může být paměťově dosti náročné. Při spouštění z příkazové řádky nám stačilo přidat argument, který zvětšil velikost využitelné paměti na Haldě. To u spustitelného jar souboru udělat nemůžeme.

Z těchto důvodů jsme vytvořili novou třídu „Spusteni“. Tato třída obsahuje pouze metodu main. Jejím jediným cílem je spustit náš původní program s požadovanými argumenty. Tato třída již nic dalšího nepotřebuje, tedy z ní můžeme jednoduše udělat náš spustitelný jar soubor.

Závěr

Cílem práce bylo vytvořit program pro hraní libovolných karetních her. Takový program jsme nakonec naimplementovali. Problémem našeho řešení je velká složitost. Sice v našem programu jde naprogramovat libovolnou hru, ale pro obyčejného uživatele je to příliš náročné. Aby si totiž uživatel mohl hru vytvořit, tak potřebuje mít základní znalosti programování. Na druhou stranu jsme si stanovili, aby univerzálnost měla přednost před jednoduchostí.

Přínosem je, že uživatel znalý programování nemusí od základu vytvářet celou hru. Stačí mu zadat pravidla a obrázky karet. Program se už sám postará o GUI, síťovou komunikaci a umělou inteligenci.

Do budoucna by bylo vhodné jazyk pro zadávání pravidel rozšířit. Důvodem je, že díky použití pouze základních programovacích prvků lze sice některé věci udělat, ale za cenu velké složitosti a ne příliš velké efektivity. Příkladem mohou být moderní karetní hry. V nich se často používá více karetních balíčků. Náš program, ale v základu počítá pouze s jedním, tedy podporu více balíčků si uživatel musí v pravidlech doprogramovat.

Dalším vhodným rozšířením by mohlo být vhodné hlášení chyb v pravidlech. My sice již máme kontrolu syntax, ale pokud nastane chyba až za chodu, tak budeme mít problém zjistit, kde k ní došlo.

Pro uživatele jsme vytvořili také dvě různé umělé inteligence. První z nich si uživatel může sám navrhnout pro konkrétní potřeby hry. Pokud nechce uživatel psát vlastní umělou inteligenci, potom může použít univerzální umělou inteligenci, která je použitelná pro všechny hry. Problémem této umělé inteligence je její ne příliš velká rychlost. Aby rychlost byla únosná, tak jsme museli UI značně omezit, tedy výsledky, které dává nemusí být v některých hrách příliš vhodné.

Seznam použité literatury

[1] RL Software: Virtual Deck

`http://www.rlsoftwares.com/baralhovirtual/`

[2] Java Platform SE 7: Swing

`http://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html`

[3] Java Platform SE 7: Net

`http://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html`

[4] Russel, Stuart J a Peter Norvig. *Artificial intelligence: a modern approach.* 3. vydání. Upper Saddle River: Prentice Hall, 2010. ISBN 978-0-13-604259-4.

[5] Java Platform SE 7: XMLEncoder

`http://docs.oracle.com/javase/7/docs/api/java/beans/XMLEncoder.html`

[6] Java Platform SE 7: XMLDecoder

`http://docs.oracle.com/javase/7/docs/api/java/beans/XMLDecoder.html`

[7] Automaty a gramatiky

`http://kti.mff.cuni.cz/~bartak/automaty/lectures/all_lectures.pdf`

Seznam použitých zkratek

IP Internet Protocol

XML Extensible Markup Language

UI Umělá Inteligence

GUI Grafické Uživatelské Rozhraní

TCP Transmission Control Protocol

Přílohy

CD, které obsahuje následující soubory:

- doc - adresář obsahující dokumetaci vygenerovanou pomocí javadoc
- gkh - adresář obsahující spustitelný program
- src - adresář obsahující zdrojové soubory
- manual.pdf - uživatelská dokumentace
- prace.pdf - tato práce