VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KNIHOVNA PRO PODPORU KARETNÍCH HER NA MOBILNÍCH ZAŘÍZENÍCH

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE VÁCLAV HODEK
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KNIHOVNA PRO PODPORU KARETNÍCH HER NA MOBILNÍCH ZAŘÍZENÍCH

LIBRARY FOR CARD GAMES ON MOBILE DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE VÁCLAV HODEK
AUTHOR

VEDOUCÍ PRÁCE ING. ZBYNĚK KŘIVKA, PH.D.
SUPERVISOR

BRNO 2008

Abstrakt

Tato bakalářská práce se zabývá vývojem knihovny pro podporu karetních her na mobilních zařízeních na platformě J2ME a také obecnými doporučeními užitečnými pro vývoj a testování. Samotná knihovna je pak navržena s ohledem na co největší podporu nejen ze strany mobilních telefonů, ale i desktopových systémů, což je výhodné zvláště při testování knihovny. Knihovna by měla být schopna řešit co nejvíce různých druhů karetních her.

Klíčová slova

J2ME, MIDP, CLDC, mobilní aplikace, mobilní hry, testování mobilních aplikací, omezení mobilních zařízení, mobilní knihovna, karetní hra, karty, karetní balíček

Abstract

This thesis deals with a development of a library used for supporting card games on mobile devices and also brings some common suggestions that can be useful during the development and testing mobile applications. The library is designed to be supported by as many mobile devices as possible and to be supported by a desktop system as well since it can be very helpful especially for testing. The library is supposed to allow a developer to handle as many kinds of card games as possible.

Keywords

J2ME, MIDP, CLDC, mobile applications, mobile games, mobile applications testing, mobile device limitations, mobile library, card game, cards, card package.

Citace

Hodek Václav: Knihovna pro podporu karetních her na mobilních zařízeních. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Knihovna pro tvorbu karetních her na mobilních zařízeních

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zbyňka Křivky, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Václav Hodek
1.5.2008

Poděkování

Rád bych poděkoval Ing. Zbyňku Křivkovi, Ph.D. pod jehož vedením jsem práci vypracoval, za četné podněty, které přispěly ke zlepšení kvality této práce.

Obsah

1 Úvod

S rozšiřujícím se vlivem mobilních telefonů se stala oblast mobilní zábavy jedním z nejprogresivněji rostoucích odvětví a kromě obvyklého obsahu, který tvoří zejména melodie, obrázky a animace, přibyl také interaktivní obsah tvořený především hrami a aplikacemi. S rostoucí oblibou her vznikla nutnost připravovat je stále atraktivnější a technicky lepší, což s mnohdy velice omezenými možnostmi hardware i software mobilního telefonu není nejsnazším úkolem.

Dalším faktorem, který má negativní vliv na vývoj mobilních her, je také mnoho různých standardů a jejich podpora ze strany jednotlivých výrobců mobilních telefonů. Úspěšný projekt musí podle očekávání fungovat na co největším počtu zařízení, a proto je mnohdy nutné napsat vybrané části odlišně podle typu zařízení, a nebo použít pouze funkce patřící do standardů, jež podporují všechna cílová zařízení. V této práci se zaměříme hlavně na druhou možnost.

Nesmíme zapomenout ani na fakt, že aplikace mnohdy nelze kvalitně ladit. Někteří z výrobců sice poskytují možnost testovat aplikace během vývoje na mobilním telefonu, který zasílá ladící výpisy do připojeného počítače, ale to je spíše výjimka. Ve většině případů se musíme spolehnout na vývojové nástroje jednotlivých výrobců, které ovšem nebývají příliš kvalitní a chování aplikace v těchto emulátorech nemusí plně odpovídat chování stejného kódu na mobilním telefonu.

Podívejme se tedy na jednu z dnes nejrozšířenějších platforem pro vývoj aplikací pro mobilní zařízení, J2ME, a seznamme se s vybranými omezeními a úskalími, se kterými se můžeme při použití tohoto implementačního prostředí setkat.

Knihovna pro tvorbu karetních her na mobilních zařízeních, která je hlavním předmětem této bakalářské práce, bude vytvořena právě v J2ME a ukážeme si na ní některé postupy vhodné pro snadný vývoj a testování přímo na počítači. Zároveň se budeme soustředit i na co nejširší podporu knihovny, co se týče jednotlivých standardů J2ME.

Abychom byli schopni vytvořit knihovnu co nejobecnější i z hlediska podpory různých typů karetních her, musíme specifikovat, co přesně je karetní hra a jaké operandy a operátory se v ní vyskytují. Analýzou různých karetních her získáme společné požadavky, ze kterých pak při vytváření knihovny budeme vycházet.

Od knihovny budeme také očekávat zajištění základní interakce s uživatelem, a proto musíme v požadavcích zohlednit nejenom operace, které lze v rámci karetní hry provádět, ale i operace, které mohou být provedeny na základě vnějších vlivů. V oblasti mobilních telefonů existují dvě základní možnosti interakce uživatele s aplikací, a to zejména pomocí klávesnice, a nebo dotykového displeje. Měli bychom zohlednit obě dvě možnosti.

Knihovna bude od začátku vytvořena jako samostatná práce a není založena na kódu ani principech třetí strany.

2 Pojmy a definice

V textu této bakalářské práce budeme pracovat s pojmy z oblasti karetních her a vývoje aplikací pro mobilní zařízení. Uveďme si definice, které se daných pojmů týkají.

Karta

Nejmenší herní jednotka v karetních hrách. Každá karta má vždy své označení a nemusí být ve hře unikátní, neboť se může vyskytovat více karet se shodným označením.

Barva

V mnoha karetních hrách je barva pojmenováním pro skupinu karet s určitým společným prvkem.

Karetní balíček

Skupina libovolného počtu karet. Balíček nemusí být tvořen žádnou kartou a pak ho nazýváme prázdný. Do balíčku lze kdykoliv přidávat další karty a z neprázdného balíčku lze karty kdykoliv odebírat.

Karetní hra

Pravidla definující podmínky přesunu karet mezi jednotlivými karetními balíčky. Hra se obvykle skládá z daného počtu karet a z daného počtu balíčků. Jednotlivým hráčům může být odepřeno manipulovat s některými z balíčků a naopak některé mohou využívat pouze oni.

Hrací stůl

Dvourozměrná plocha, na které se nacházejí karetní balíčky. Balíčky mohou být umístěny na této ploše na libovolných souřadnicích. Ne všechny balíčky musí být nutně viditelné.

Aplikace

Spustitelná aplikace ve vhodném formátu pro mobilní zařízení nebo operační systém, jejímž účelem je realizovat danou karetní hru.

Platforma

Prostředí, ve kterém bude aplikace provozována. Může se jednat o různé operační systémy nebo o programové vybavení mobilních telefonů určené k provozu aplikací.

Midlet

Aplikace určená pro spuštění na mobilním telefonu vytvořená v micro edici Javy.

Knihovna

Souhrn funkcí, tříd a dalších objektů určených k řešení specifického úkolu. Obvykle zahrnuje často užívané techniky, které se opakují v různých aplikacích.

Portace

Verze midletu určená pro specifickou skupinu zařízení. Vzniká tzv. portováním, což je úprava midletu a jeho přizpůsobení konkrétní skupině zařízení.

3 Historie karetních her

V Evropě jsou karetní hry známé již více než 600 let a od svého vzniku byly velice populární mezi všemi společenskými a sociálními vrstvami.

Staré evropské karty často zobrazují boha Merkura, protože podle jedné z legend, kterými jsou karty opředeny, jsou Merkurovým darem. Podle jiné legendy vznikly karetní hry k ukrácení dlouhé chvíle při obléhání Tróje, a nebo je jejich původ připisován antickému filozofovi Chilonovi, který je údajně navrhl jako prostředek pro chudé, aby zapomněli na svoji bídnou životní situaci.

Přestože podle některých teorií je vznik karetních her situován do Indie, Japonska, Koreje či Persie, vznikly pravděpodobně v Číně, odkud také pochází první podložené zmínky o hraní karet. V té době se nejednalo o karty v dnešním slova smyslu, ale spíše o kombinaci karet a hry domino, a podle dochovaných kronik hrál sám čínský císař Mu-tsung roku 969 n.l. na Nový rok karty se svými ženami.

Karty podobné těm dnešním se začaly objevovat později a mezi první z nich patřily pravděpodobně karty Hakka Pai, které se řadí do skupiny tzv. peněžních karet. Existují teorie, že dnešní karty se vyvinuly do své nynější podoby právě z peněz.

Podobně jako je nejasnostmi zastřen samotný původ karet, není zcela jasné ani to, jak se karty dostaly do Evropy. Teorie o tom, že je přinesl Marco Polo, křižáci nebo cikáni, byli vyvráceny a vzhledem k nedostatku informací nelze potvrdit žádnou domněnku o jejich průniku na náš kontinent.

První zmínka o kartách v Evropě pochází z roku 1371 z Katalánska a ještě před rokem 1380 se objevují další zmínky z mnoha významných evropských států. Bohužel se ve většina dochovaných informací týká zákazu hraní karet v dané zemi, a proto nemohou tyto zmínky napomoci určení původu karet a jejich rozšíření v Evropě.

Více informací o kartách se dozvídáme až ve 14. a 15. století, kdy se v Itálii rozšířila karetní hra pojmenovaná *naibi* a ve Španělsku *naypes*. Slovo *naibi* je s největší pravděpodobností odvozeno z arabského *laib*, což v doslovném překladu znamená hra a nebo také hračka. Nabízí se i možnost, že *naibi* je odvozeno z jiného arabského slova, a to *nabi*, které znamená prorok. Můžeme se tedy domnívat, že tento název je spojen s věštěním z karet. Další vysvětlení vycházejí z arabsko-hebrejské terminologie, přičemž nejvíce je příbuzný arabský výraz *naib*, jenž znamená doslova zástupce. Je tedy vidět, že na vývoj evropských karet působily silné arabské vlivy.

Nejstarší dochovaná sada karet pochází z 15. století. Jedná se o arabské karty a za zemi jejich původu je považována oblast dnešního Egypta. Arabské barvy karet měli velice blízko k italským a částečně i španělským. Setkáváme se na nich s barvami *Darahim* (mince), *Tuman* (poháry), *Suyuf* (meče) a *Jawkan* (hole na pólo).

Za základ pojmu karta jsou považovány středověké termíny *chartae* nebo *cartae* a nebo také *charticelae*, které označovaly papír vyrobený z papyru. Označení se pak pravděpodobně přeneslo a vyvinulo v dnešní termín *karta*.
[1]

4 Požadavky karetních her

Karetní hrou rozumíme souhrn pravidel pro přesun karet mezi karetními balíčky. Hra se obvykle skládá z určitého počtu karet a z určitého počtu balíčků, ale nemusí to tak být vždy. Zatímco celkový počet karet se většinou nemění, avšak ani to není pravidlem, počet balíčků nemusí být v průběhu hry konstantní.

Pod pojmem balíček si představme jakoukoliv skupinu karet. Tato skupina nemusí obsahovat žádnou kartu a budeme ji pak nazývat prázdným balíčkem. Pokud tedy hovoříme o přesunu karet, máme na mysli vždy odstranění karty z jednoho balíčku a její vložení do balíčku jiného. Cílový balíček může pro tento účel vzniknout a naopak ten, z něhož kartu přesouváme, může být po přesunu odstraněn.

Jednotlivým hráčům může být odepřeno manipulovat s některými z balíčků a naopak některé mohou využívat pouze oni. Manipulací přitom rozumíme, že hráč může na základě pravidel a svého úsudku rozhodnout, kterou kartu a kam přesune.

4.1 Karty

Jedním z hlavních požadavků na knihovnu je možnost použít ji pro realizaci co nejširšího spektra karetních her. Splnit tento požadavek znamená pracovat s různými počty karet různých barev, a přestože je většina karetních her založena na německých 32-listových nebo francouzských 52-listových kartách [2], měli bychom vzít v úvahu také karetní hry, kde nemusí být karty děleny do typických barev. Jako příklad uveďme hry ve stylu Magic The Gathering [3] a Doomtrooper [4]. Existují také hry založené z části na klasických kartách, jež jsou doplněny o karty speciálního významu. Tak je tomu například u hry Taroky [5], do této skupiny však může spadat i karta *žolík*.

Abychom mohli nabídnout efektivní nástroj pro práci s kartami, musíme být schopni dělit je do skupin na minimálně jedné úrovni. Z definice pojmu barva je zřejmé, že tento výraz označuje právě příslušnost k určité skupině. A přesně takové skupiny mohou být výhodné nejen u klasických karet, ale i u zmíněných her používajících jiný typ karet.

Kromě příslušnosti ke skupině musí mít každá karta i své primární označení. U klasických karet se označuje pojmem hodnota. Berme v úvahu různé typy her a umožněme uživateli knihovny použít primární označení dle svého nejlepšího uvážení.

Každá z karet je identifikována kombinací skupiny a primárního označení. Karet se shodnou skupinou i primárním označením se může vyskytovat v jedné hře či balíčku více.

4.2 Karetní balíčky

Na základě uvedených údajů je tedy zřejmé, že chceme-li vytvořit jakoukoliv karetní hru, musí nejdříve existovat mechanismus pro definování balíčků. Jednotlivé balíčky mohou mít různé parametry, a to zejména v oblasti vlastní vizuální prezentace a vizuální prezentace obsažených karet. Celkem můžeme provést rozdělení na tři druhy, které budou uvedeny níže. Rozdělení je přitom výhodné hned z několika důvodů.

Při práci s daným typem balíčku máme k dispozici množinu všech nad ním existujících operací. Neexistují nerelevantní operace, které by v danou chvíli neměly efekt.

Nevzniká nutnost ukládat pro daný typ balíčku data, která by přímo nesouvisela s jeho funkcí, čímž dochází k úspoře paměti.

Vhodným vytvořením objektového modelu, reprezentujícího tyto balíčky, necháváme prostor pro uživatelskou definici odvozených tříd se speciálním významem, které dědí jen požadovaný rozsah funkcí.

Vyhneme se duplicitní implementaci funkcí se stejnou sémantikou.

4.2.1 Balíčky sloužící k uchování karet

Ne všechny karty ve hře musí být vždy nutně použity a ty, které použity nejsou, nemusí být součástí hracího stolu, např. z důvodu úspory místa. Mohou existovat i další důvody, proč je výhodné karty uchovávat, aniž by byly vizuálně prezentovány. Pro tento účel existuje skrytý balíček, který slouží jako uložiště karet a nemusí disponovat žádnými dalšími funkcemi.

4.2.2 Vizuální neinteraktivní balíček

Většinou budeme požadovat, aby byl balíček a jeho obsah viditelný. Pokud se nejedná o balíček, u kterého se předpokládá možná manipulace ze strany hráče, vystačíme si pouze s neinteraktivním balíčkem, jenž snadno získáme, rozšíříme-li výše zmíněný balíček, sloužící k uchování karet, o vykreslování.

Karty umístěné v balíčku a vykreslované spolu s ním mohou být zobrazeny podle svých vlastních nastavení. Tedy mohou určit, zda jsou vidět pro hráče lícem nebo rubem. V mnoha případech však bude výhodnější nabídnout možnost definovat společné chování pro celý balíček. Z analýzy typický her je pak zřejmé, že nejčastěji se vyskytuje možnost, kdy všechny karty sdílejí stejné nastavení, a nebo možnost, kdy všechny kromě vrchní karty sdílejí nastavení a viditelnost vrchní karty je opačná.

4.2.3 Interaktivní balíček

Interaktivní balíček disponuje všemi možnostmi vizuálního neinteraktivního balíčku a rozšiřuje ho o reakci na vstup od uživatele. Vizuální prezentace balíčku bude odlišná, protože je potřeba lépe pracovat se stavy, kdy nemohou být najednou zobrazeny všechny karty, například pro nedostatek prostoru. I kapacita hráčovy ruky v reálném životě je omezena.

Tento typ balíčku poskytuje uživateli možnost výběru jedné z obsažených karet, a to prostřednictvím jakéhokoliv vstupu daného zařízení. Vybraná karta bude odlišně vizuálně prezentována a uživatel může potvrdit svůj výběr, a provést s ní relevantní operaci.

4.3 Definice pravidel

Z teoretického hlediska je možné definovat pravidla karetní hry několika způsoby. Uveďme si alespoň některé z nich, nad nimiž jsem během zpracovávání bakalářské práce uvažoval a snažil se najít jejich výhody a nevýhody. Názvy uvedených metod nejsou standardizované a způsoby definice pravidel jsem pojmenoval tak, aby co nejvíce vystihovaly zamýšlený princip.

Na první pohled je zřejmé, že všechny metody se zabývají hlavně definicí podmínek, na základě kterých mohou být provedeny akce hráče. Pravidla všech karetních her jsou souborem restriktivních podmínek a důsledků provedených akcí. Zatímco akce se mohou diametrálně lišit hru od hry, podmínky mají společné charakteristiky, díky nimž lze uvažovat nad unifikovaným postupem jejich definice a zpracování. Měli bychom přitom brát ohled na přehlednost zápisu a možnost separace od hlavního kódu aplikace a také na rychlost zpracování.

4.3.1 Prerekvizitní podmínky

Princip definice pravidel, inspirovaný aspektově orientovaným programováním, spočívá ve vytvoření prerekvizitních podmínek.

Uvažujme mechanismus, jenž umožní programátorovi definovat všechny možné podmínky, které tvoří prerekvizity pro provedení akce. Knihovna se pak postará o prověření všech těchto podmínek a předá vnějšímu kódu pouze požadavek na provedení akce, který splnil všechny předdefinované předpisy. Tím jsou pravidla hry definována pomocí zmíněných podmínek. Provedení akce je sice pro funkčnost aplikace nezbytné, ale souvislost s vlastními pravidly karetní hry není tak markantní jako ověření její proveditelnosti.

Z pohledu aplikace je kód mnohem přehlednější, protože velká část ověření, podmínek a pravidel může být delegována na knihovnu samotnou a jejich zápis může být od zbytku kódu separován.

Právě separace však může být problémem u her, kde je potřeba pracovat s mnoha různými stavy a kde vzniká potřeba předávat mezi vlastní aplikací a řešením podmínek velké množství

různých parametrů. Pokud se hra sestává z velkého množství balíčků a existují složité vazby, musíme mít možnost pracovat se všemi objekty, a proto, abychom vytvořili knihovnu skutečně obecnou, museli bychom ponechat jejímu uživateli možnost provádět nad každým z herních objektů všechny operace, které lze na takový objekt aplikovat. Tím se ovšem dostáváme k jinému principu, který by byl obecně vhodnější a který si uvedeme níže.

Aparát pro definici prerekvizitních podmínek by tak mohl být velice užitečným rozšířením knihovny při budoucím vývoji, z pohledu implementace čistě obecného řešení však není nejvhodnější cestou. Jak bude zmíněno i v dalších kapitolách, implementací takového rozšíření by se zvýšila především velikost knihovny, a to v oblasti mobilních aplikací není žádoucí efekt.

Předpokládejme, že v dané karetní hře lze z hráčova balíčku odhodit kartu do odhazovacího balíčku pouze tehdy, pokud má shodnou barvu nebo hodnotu jako karta na vrchu odhazovacího balíčku. Podívejme se na příklad zápisu pravidla pomocí prerekvizitních podmínek. Uvedený kód je pouze ilustrativní a je zapsán v jazyce Java.

```
// definice prerekvizitnich podminek
int simpleCondition = game.createNewConditionGroup();
game.addConditionToGroup(simpleCondition,
     "user.topcard.suit=throw.topcard.suit");
game.addOrConditionToGroup(simpleCondition,
     "user.topcard.rank= throw.topcard.rank");

// pozadavek na provedeni akce
userPack.knownAs("user");
throwPack.knownAs("throw");
game.letUserMoveCard(userPack, throwPack, simpleCondition);
```

Jiným přístupem by mohlo být vytvoření třídy pro řešení prerekvizitních podmínek odvozené z abstraktního předka. Opět si ukažme ilustrativní úsek kódu.

```
// odvozena trida pro reseni prerekvizitni podminky
public class Simple extends PreCondition {
  public boolean solve(Package pack1, Package pack2) {
    ....
  }
}

game.letUserMoveCard(userPack, throwPack, new Simple());
```

4.3.2 Validace stavu

Opakem prerekvizitních podmínek je princip validace stavu, který je založen na kontrole situace, jež nastane po provedení akce. Na základě ověření je nový stav potvrzen a ponechán, a nebo odmítnut, a pak musí být provedena jiná akce. Existují hry, kdy je kontrola konečného stavu výhodnější. Například pokud je operace složitá a může manipulovat s mnoha herními objekty, vyžádal by si princip prerekvizitních podmínek kontrolu mnoha faktorů.

Platí zde prakticky stejné výhody a nevýhody jako u prerekvizitních podmínek. Kód aplikace je přehlednější, avšak pro obecné řešení je potřeba nabídnout sortiment všech možných operací.

Velmi zajímavá by byla kombinace tohoto a předchozího uvedeného principu. Ověření by v takovém případě mohlo být rozděleno do dvou jednodušších částí.

Princip validace stavu hry je zajímavou možností, jak knihovnu rozšířit do budoucna.

Předpokládejme ilustrativní situaci, kdy se očekává, že hráč doplní vždy kartu tak, aby nenarušil postupku. Na barvě karty přitom nezáleží.

```
// odvozena trida pro overeni stavu hry
public class Check extends StateVerifier {
  public boolean checkState(Package target) {
    // Serad balicek podle hodnoty vzestupne
    target.sort(SORT_BY_RANK, SORT_ASC);
    // Zkontroluj, ze predchozi karta je vždy o jednu
    // mensi nez karta nasledujici, kdyz ne vrat false
    for(int i=1;i<target.getCardCount();i++) {
      if (target.card[i - 1].getRank() ==
          target.card[i].getRank() - 1) return false;
    return true;
  }
}

...

game.letUserMoveCard(packUser, packTarget, new Check());
```

4.3.3 Plnou kontrolou

Jak již napovídá název principu, umožníme uživateli knihovny manipulovat se všemi objekty bez omezení a poskytneme mu všechny operace, které by mohl eventuelně použít.

V obou předchozích principech je zřejmý nedostatek v tom, že prakticky celou kontrolu nad operací přebírá knihovna samotná. V mnoha situacích je to velmi výhodné, ale existují i situace,

kdy je potřeba zareagovat odlišně. V takovou chvíli bychom potřebovali další mechanismy pro přerušení akce. Může nastat i situace, kdy je potřeba provádět operace strojově.

Chceme-li implementovat knihovnu na co nejobecnější úrovni, je nejlepší volbou právě tento princip. Není vhodný z hlediska přehlednosti kódu a může vést k velmi složitým konstrukcím, v nichž snadněji vznikne chyba, ale jeho vyjadřovací schopnosti jsou větší než u předchozích principů a v konečném důsledku je dostatečným aparátem pro implementaci dalších definičních možností.

Tento přístup je výhodný i z hlediska velikosti knihovny. Jednotlivé objekty postačí rozšířit o potřebné operace.

4.4 Operace a operandy

V karetních hrách existují dva základní druhy objektů, se kterými lze pracovat. Jsou jimi karty a balíčky.

Nad kartami jako takovými nelze provádět prakticky žádné operace. Výjimkou je vytváření karet a označení karty z důvodu jejího zvýraznění pro případ, že na ni chceme upozornit změnou vizuální prezentace. Označování vybraných karet je nezbytné pro realizaci interaktivního balíčku.

Přesto hrají karty v jednotlivých operacích významnou roli, protože se jich vždy účastní. Operace nad balíčky poskytují mechanismus, jak s obsahem balíčku a tedy s kartami pracovat. Operace lze provádět v rámci jednoho balíčku a pak se jedná o vložení karty do balíčku, změnu pořadí karet, řazení karet a získání vybrané karty.

Karty lze přesouvat mezi dvěma a více balíčky. Přesouvaná karta je z výchozího balíčku odstraněna a vložena do nového. Lze vyjmout jakoukoliv kartu a vložit ji na vrchol cílového balíčku. To je typické chování, které bude vyhovovat většině karetních her, a pro případ, že by nevyhovovalo, lze pořadí karet změnit.

4.5 Analýza požadavků

Knihovna bude řešit vytváření karty. Tento mechanismus poskytneme pouze na jednom místě, a to prostřednictvím hlavního herního balíčku, aby bylo vytváření karet centralizované, a zejména aby se nekomplikovaly ostatní herní objekty funkcemi, které nebudou při samotné hře využívány.

Dalším požadavkem je, aby knihovna disponovala třemi základními druhy balíčků. Jeden z nich bude sloužit pouze k uchování karet a nebude mít žádnou vizuální prezentaci. Druhý typ balíčku bude řešit i vykreslování a posledním typem, nejdůležitějším pro vytvoření hry, bude balíček interaktivní, jenž zajistí reakci na vstupy jednotlivých hráčů.

Od karet očekáváme, že kromě reprezentace své barvy a hodnoty dovolí měnit individuálně stav viditelnosti, tj. zda budou zobrazeny lícem nebo rubem. Stav viditelnosti může být také ovlivněn balíčkem. Pro účely zobrazení lze karty zvýraznit. Výhodné budou dva režimy. Jeden z nich nazveme

aktivní. Bude sloužit zejména interaktivnímu balíčku k označení karty, kterou má uživatel právě vybranou. Druhým režimem bude běžné zvýraznění karet. Zatímco aktivní režim může být ovlivněn ze strany knihovny, běžné zvýraznění bude sloužit uživateli a zůstane plně v jeho kompetenci.

Uživateli knihovny nabídneme možnost přesouvat karty z vrchu nebo spodu jednoho balíčku do druhého. Pro potřeby většiny her bude výhodný i přesun náhodně vybraných karet.

Pro každý balíček bude možné zjistit několik informací. Užitečný je počet obsažených karet a přístup k jednotlivým kartám.

Existují karetní hry, kdy může hráč používat více interaktivních balíčků během jednoho svého tahu. Kromě vlastní navigace uvnitř balíčku poskytne knihovna možnost navigace mezi balíčky. Není žádoucí, aby měl hráč možnost se přepnout na balíčky, s nimiž není oprávněn disponovat. Zabráníme tomu výběrem aktivovatelných balíčků.

5 Implementační prostředí

Vzhledem k tomu, že chceme, aby využitelnost knihovny byla co nejširší, zvolíme jako implementační prostředí J2ME (Java 2 MicroEdition), které je dnes dostupné na největším procentu mobilních zařízení.

5.1 J2ME

J2ME (Java Platform, Micro Edition) je aplikační prostředí vytvořené společností Sun Microsystems za účelem poskytnout robustní a flexibilní nástroj pro tvorbu aplikací na mobilních zařízeních, PDA, set-top boxech, tiskárnách a dalších vestavěných systémech. J2ME zahrnuje uživatelské rozhraní, bezpečnostní model, vestavěnou podporu síťových protokolů a podporu pro síťové i offline aplikace, které mohou být stahovány dynamicky. Vychází z jednoho z nejrozšířenějších jazyků pro obecné použití – Java – a je plně kompatibilní s verzí 1.3 zdrojového kódu. Pro kompilaci lze tedy využít běžný kompilátor. [6]

J2ME se od standardní a enterprise edice liší hlavně rozsahem dostupných knihoven a starší standardy navíc i absencí práce s desetinnými čísly. Jsou ovšem zavedeny knihovny nové, obsahující funkce, které přímo vycházejí z požadavku na tvorbu aplikací pro mobilní zařízení.

Micro Edition se od standardní a enterprise edice liší také v přístupu k bezpečnostní politice. Značně omezené možnosti mobilních zařízení nedovolují využít stejného principu jako u PC, a proto je nutné aplikace preverifikovat. Během tohoto procesu se provede kontrola některých bezpečnostních omezení a informace se zapíšou přímo do zkompilovaného bytecodu aplikace. Při spuštění a běhu aplikace na mobilním zařízení jsou otázky bezpečnosti řešeny mnohonásobně rychleji.

J2ME je technologie složená z několika menších částí, tzv. API (Application Programming Interface), rozdělených na konfigurace, profily a volitelné balíčky.

Konfigurací rozumíme základní množinu knihoven a funkcí virtuálního stroje, které by měly být přítomné v každé implementaci J2ME. Při spojení konfigurace s alespoň jedním profilem získáme dostatečné prostředí pro vývoj aplikací pro danou platformu. [7]

Zmíněné profily jsou množinou standardních API, která úzce souvisejí vybranou skupinou mobilních zařízení a poskytují funkce, jež mohou být i velice specifické pro daný typ zařízení. Profil musí být vždy podporován danou konfigurací. [8]

Java aplikace určená pro platformu J2ME se obvykle nazývá midlet.

5.2 CLDC

CLDC (Connected Limited Device Configuration) je nejrozšířenější konfigurací pro mobilní telefony. Existují i další konfigurace, ale zdaleka nejpoužívanější je právě CLDC. Obecně se jedná o konfiguraci zaměřenou především na hardwarově slabší zařízení, typicky s 16-ti nebo 32-ti bitovým procesorem a s prostředím s dostupnou pamětí od 160kB do 512kB. Právě proto jsou jí vybaveny především mobilní telefony a některá PDA. Počítá se rovněž s možností limitovaného připojení k některému druhu sítě a také s nízkou spotřebou a s provozem převážně na baterie. Konfigurace CLDC řeší bezpečnostní politiku a záležitosti spojené s procesem preverifikace. [7]

Konfigurace CLDC existuje v současné době ve dvou verzích. Novější, s označením 1.1, je zpětně kompatibilní a přidává k verzi 1.0 některé nové vlastnosti, které budou uvedeny níže. Kromě rozšíření dostupných funkcí také definuje o něco větší požadavky na hardware zařízení, především na paměť, jejíž minimum se vzhledem k rozšířením zvýšilo na 192 kB. [9]

Konfigurace CLDC 1.0 obsahuje především část základních knihoven java.lang a java.util, částečnou podporu pro vstupní a výstupní datové proudy z balíčku java.io a základní podporu limitovaného připojení k síti, která je obsažena v nově zavedeném balíčku javax.microedition.io. [7]

Ve verzi CLDC 1.1 bylo provedeno několik úprav již existujících objektů tak, aby se více přiblížily jejich protějšku ve standardní edici Javy. Byla také přidána podpora desetinných čísel a slabých referencí. [9]

5.3 MIDP

MIDP (Mobile Information Device Profile) je profil podporovaný konfigurací CLDC a stejně jako CLDC je nejrozšířenějším profilem v oblasti mobilních zařízení. [8]

Podobně jako CLDC definuje hardwarové a softwarové požadavky na virtuální stroj J2ME. V oblasti hardware jsou to pak především možnosti vstupu a výstupu. Pro výstup se předpokládá displej s rozlišením nejméně 96x54 s 1-bitovou barevnou hloubkou, poměr rozměrů pixelu 1:1. Vstupním zařízením pak může být buď klávesnice určená pro ovládání jednou rukou, klávesnice určená pro ovládání oběma rukama nebo dotykový displej. Ve specifikaci jsou zahrnuty i paměťové požadavky, zvláště pak pro komponenty MIDP, persistentní data a běhové proměnné. [8]

Po stránce software definuje MIDP následující požadavky:

- Základní jádro pro práci s hardwarovou vrstvou.
- Mechanismus pro řízení životního cyklu midletu.
- Mechanismus pro práci s persistentními daty.
- Přístup pro zápis a čtení z bezdrátového spojení.
- Mechanismus pro práci s časem, časovými známkami a s časem persistentních dat.
- Minimální možnosti práce s grafickým výstupem a uživatelským rozhraním.

- Mechanismus pro zachytávání vstupu od uživatele.
- Rozšíření bezpečnostní politiky CLDC.

Existuje více verzí MIDP profilu. První verzí byl profil MIDP 1.0, který poskytoval většinu potřebných prostředků pro vývoj aplikací, avšak měl omezenou bezpečnostní politiku a velmi omezené multimediální možnosti. Proto vznikl profil MIDP 2.0. Zavedl rozšíření bezpečnostni dělením aplikací do tzv. domén, push registry a rovněž přidal multimediální rozšíření a specifická rozšíření pro hry. Zavedeny byly také sockety. [10]

Vyžadovány jsou větší požadavky na hardware, zejména na paměť, a jsou definovány zvukové formáty, bezpečnostní protokoly a kódování národních znaků, které musí zařízení podporovat. [10]

Profil MIDP 2.0 je plně zpětně kompatibilní s profilem MIDP 1.0.

V dnešní době již existují některé telefony společnosti Nokia s implementací MIDP 2.1. Profil MIDP 2.1 nepřináší téměř nic nového a pouze reviduje profil MIDP 2.0. Některé z volitelných funkcí MIDP 2.0 se staly striktně vyžadovanými. Profil MIDP 2.1 je podporován pouze konfigurací CLDC 1.1, a tím tedy zanikl problém výběru vhodné konfigurace a profilu. [11]

5.4 Volitelné balíčky

Při svém vzniku bylo J2ME tvořeno pouze jednou konfigurací a jedním profilem. Množina funkcí poskytovaná těmito standardy, byla značně omezená, a začaly vznikat tzv. volitelné balíčky. Původní představa byla taková, že s evolucí cílových zařízení budou vznikat nové profily, ale podpory různých specifik integrovaných pouze do některých zařízení si vyžádaly jiný přístup. Dnes je velká část běžně užívaných funkcí definována ve volitelných balíčcích a mnoho z nich se stalo vedle CLDC a MIDP standardním vybavením mobilních zařízení. [12]

Volitelné balíčky nejsou vyžadovány pro žádný typ mobilního telefonu, a jejich podpora je tedy plně závislá na rozhodnutí výrobce. Už z tohoto důvodu nejsou nejvhodnější při tvorbě obecných knihoven, pokud ovšem daná knihovna není založena právě na některé z jejich funkcí.

V současné době existuje více než 40 volitelných balíčků, které jsou v různém stádiu definice. Mezi známější patří zejména: [13]

- Multimediální rozšíření
- Práce s bluetooth
- Příjem a odesílání zpráv
- Volitelný balíček pro PDA
- Rozšíření pro lokalizaci
- Podpora 3D grafiky
- Práce s 2D vektory
- a další

Jak již bylo zmíněno, je lepší se volitelným balíčkům vyhnout, přesto nesmíme na jejich existenci zapomínat. Skutečně obecná knihovna by měla poskytnout mechanismus, který by umožnil nahradit některé ze standardních funkcí pomocí volitelných balíčků. Vztáhneme-li to na knihovnu pro tvorbu karetních her na mobilních zařízeních, měli bychom myslet i na transparentní způsob, který by umožnil například vytvořit celou hru za použití 3D grafiky a tedy příslušného volitelného balíčku.

5.5 Specifická API výrobců

Minimální množina možností, které J2ME nabízelo při svém vzniku, vedla výrobce k návrhu a implementaci vlastních volitelných balíčků, které ovšem nebyly navrženy jako všeobecný standard, a proto se staly většinou doménou pouze pro mobilní zařízení daného výrobce, což přináší při vývoji mobilních aplikací další komplikace.

Při návrhu knihovny bychom měli myslet i na specifická API výrobců podobně, jako je potřeba si dát pozor na volitelné balíčky. Problém s API výrobců je ten, že některá z nich řeší funkce, které by měli být v kompetenci profilu MIDP, ale buď nejsou, nebo je řešení pomocí API výrobce efektivnější, úspornější, apod. Mohou být úzce spojeny s vlastní funkcí aplikace a mnohdy jsou jednotlivé části aplikace vytvořeny několikrát pomocí různých API, aby byla zajištěna co nejširší podpora ze strany různých mobilních telefonů.

Zvažme, jakým způsobem se rozhodneme realizovat požadované funkce. Nejde pouze o to umožnit uživateli knihovny použít dané API, ale je potřeba brát v úvahu i situaci, kdy by se mohlo API objevit jako klíčová komponenta aplikace.

Mezi nejčastější specifická API výrobců patří zejména: [14]

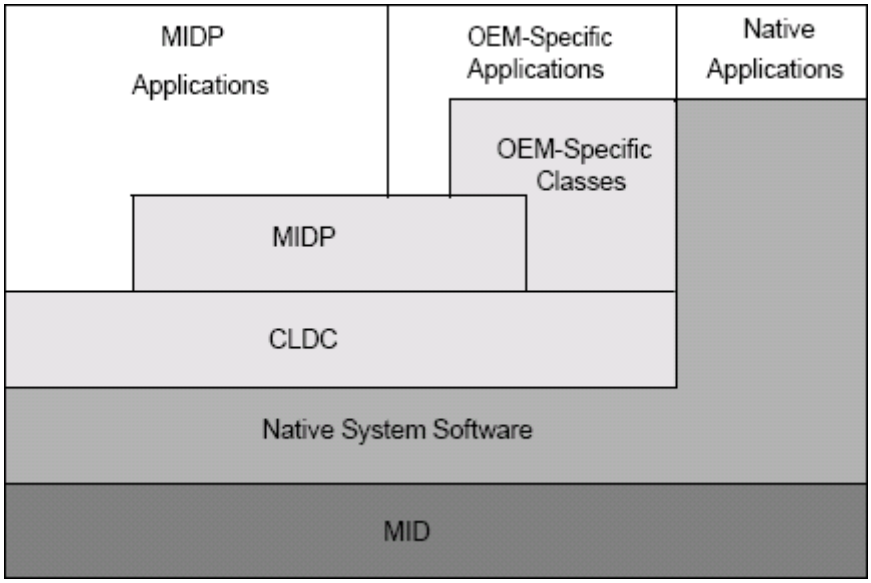
- Nokia UI API (velmi často používané)
- Siemens API
- Vodafone Service Class Library
- Motorola API
- Samsung API
- a další

5.6 Architektura J2ME

Podívejme se nyní, jak vypadá celá architektura platformy J2ME, respektive celého zařízení. Mobilní zařízení, zde MID (Mobile Information Device), má vlastní softwarové vybavení, nad kterým může být implementována konfigurace CLDC. Nad CLDC pak existuje profil MIDP a případně další volitelné balíčky.

Na obrázku je vidět i existence a vztah mezi standardními součástmi J2ME a mezi specifickými rozšířeními jednotlivých výrobců.

Oddělení softwarového vybavení MID a platformy J2ME umožňuje implementovat J2ME na zařízeních s operačním systémem i bez něj. [8]



Obrázek 5.6.1: Architektura mobilního zařízení s podporou platformy J2ME

5.7 Analýza požadavků

Seznámili jsme se se základy platformy J2ME, a můžeme tedy zvolit implementační prostředí tak, abychom si byli jisti, že všechny potřebné funkce budeme moci realizovat a že neuváženě neomezíme množinu podporovaných telefonů. Víme, jaké funkce od knihovny očekáváme, a co budeme pravděpodobně potřebovat k jejich realizaci.

Zdrojový kód budeme zapisovat kompatibilně s verzí 1.3, a získáme tak nejen možnost spustit ho na mobilním zařízení, ale pokud se vyhneme využití specifických funkcí, budeme schopni knihovnu provozovat i ve standard, případně enterprise edici Javy.

Potřebuje pracovat s abstraktními datovými strukturami, jenž jsou součástí standardních knihoven Javy a část z nich je i součástí konfigurace CLDC. Omezme se pouze na ty, které jsou dostupné v konfiguraci CLDC.

Přímý vstup a výstup není požadavkem na knihovnu. Musíme poskytnout možnost vstupu, ale vzhledem k tomu, že kódy kláves se u různých zařízení liší, realizujeme ji pomocí virtuálních kódů.

Podobně jako vstup, vyřešíme i výstup a umožníme aplikaci, která knihovnu používá, předávat skrze knihovnu referenci na libovolný grafický objekt, kontext, apod. Opět se tím vyhneme užití platformě závislého kódu a navíc umožníme použít pro výstup volitelné balíčky nebo specifická rozšíření výrobce.

Knihovnu a aplikaci můžeme propojit pomocí jednoduchého listeneru reagujícího na požadavky knihovny. Ten požádá aplikaci, tedy vnější kód mimo knihovnu, o vykreslení určitého herního objektu, apod. Výhodná je také nezávislost knihovny na vzhledu vykreslovaných objektů. Aplikace může definovat, jak by měly objekty vypadat a v průběhu svého běhu může tento vzhled kdykoliv ovlivnit.

Z uvedených informací je tedy zřejmé, že si pro celou knihovnu vystačíme se standardními knihovnami Javy, které jsou i součástí konfigurace CLDC 1.0. Celý vývoj, testování a ladění tak můžeme provést přímo v prostředí standardní edice Javy, kde máme k dispozici podstatně lepší nástroje.

Vytvořená knihovna pak bude plně přenositelná jak mezi jednotlivými typy mobilních telefonů s podporou J2ME, tak i na desktopové nebo serverové systémy.

6 Omezení a problematika mobilních zařízení

Vzhledem k velkým rozdílům mezi jednotlivými zařízeními si uveďme faktory, které mohou ovlivnit zpracování knihovny. Nesmíme zapomenout, že knihovna má poskytovat dostatečné možnosti pro podporu co nejširšího spektra mobilních zařízení a proto je potřeba poskytnout flexibilní model užití, jenž neomezí portování hry.

6.1 Výstup

Většina dnešních mobilních zařízení je vybavena displejem s poměrně nízkým rozlišením. Navíc displeje starší modelů trpí dalšími nedostatky, mezi něž patří zejména omezený počet barev, špatný kontrast a čitelnost informací. Velmi problematické jsou také pasivní displeje. Překreslování je pomalé a vznikají stíny, které ještě umocňují špatnou čitelnost.

U některých výrobců nebo modelů mobilních telefonů jsou také nejednoznačná rozlišení displeje. Například telefony společnosti Motorola mají v horní části displeje tzv. servisní část, která je neustále viditelná. Výrobcem uváděné rozlišení se tak liší od toho pro hru dostupného.

Z uvedených důvodu je potřeba nabídnout uživateli knihovny možnost definovat velmi přesně rozmístění a vzhled herních objektů, aby byl schopen vypořádat se s problematikou různých rozlišení, počtu barev a kontrastu.

Uvedené problémy se sice nejvíce týkají starších modelů, ale právě několik let staré telefony tvoří nejčastěji rozšířenou skupinu mobilních zařízení na trhu. Omezit se pouze na několik nových, špičkových telefonů by zvláště u knihovny nebylo žádoucí.

6.2 Vstup

Nejčastějším vstupem mobilních zařízení jsou různé druhy klávesnic. Méně často se vyskytují dotykové displeje. U zařízení s dotykovým displejem je obvyklá i klávesnice, ale nelze se na její existenci či použití spolehnout. Například starší komunikátory společnosti Sony Ericsson mají k dispozici buď klávesnici, nebo dotykový displej. U zařízení s dotykovým displejem bývají obvyklé také softwarové klávesnice.

Klávesnice mobilních zařízení mají ve většině případů pouze numerickou část, křížek a hvězdičku. Často se také vyskytuje tzv. D-Pad, joystick nebo kurzorový navigátor. Další klávesy jsou vyloženě závislé na konkrétním modelu a výrobci.

Přestože alespoň numerické klávesy, křízek a hvězdička mají vždy stejné kódy, nemůžeme se na ně spolehnout, protože musíme brát v úvahu i modely, kdy je rozmístění kláves jiné. To platí pro telefony s celou alfanumerickou klávesnicí a pro některé výstřední modely, jakým je například Nokia 3650.

Softwarové klávesnice jsou implementovány na úrovni operačního systému mobilního telefonu a vstup od nich není potřeba speciálně ošetřovat. Chovají se stejně jako klávesnice hardwarové. Jejich rozložení však může být i velmi odlišné. Opět si jako příklad uveďme komunikátor společnosti Sony Ericsson, u kterého je rozložení kláves softwarové klávesnice nestandardní. Klávesy jsou ve třech řadách nad sebou, nikoliv ve čtyřech, jak je běžné.

V ovládání her pro mobilní zařízení existuje v současné době trend omezující počet nutných ovládacích prvků. Velmi oblíbenými se staly zejména hry ovládané pouze jednou nebo dvěma klávesami. Při návrhu hry nebo obecné knihovny je potřeba brát na současné trendy ohled a navrhovat ovládání či interakci s uživatelem vhodným způsobem.

6.3 Výkon, velikost aplikace a paměť

Většina novějších mobilních zařízení má dostatečný výkon i k provozu složitějších aplikací, ale, jak již bylo zmíněno výše, musíme brát v úvahu i starší telefony, kde je situace o poznání horší. Mnoho z nich je naprosto nevhodných pro provoz náročnějších aplikací a je potřeba výrazných optimalizací nebo kompromisů.

Obecně nejvíce výkonu spotřebovává vykreslování a složité výpočty složené z více vnořených cyklů. V úvahu je ovšem potřeba brát i mnohočetná volání funkcí, a zvláště v často vykonávaných úsecích kódu musíme zvolit vhodný kompromis mezi výpočty nebo užitím většího množství paměti pro předpočítaná data.

Zmínili jsme se o nutnosti výrazných optimalizací. Nejčastějším řešením je omezení opakovaných výpočtů, případně úplné omezení složitějších výpočtů, kde je konečná množina determinovatelných výsledků. Takové řešení si ovšem vyžádá zvýšenou spotřebu paměťových prostředků, a právě paměť bývá u starších modelů výrazně omezena.

Na vrub starším modelům musíme připsat ještě jednu špatnou vlastnost, která se ovšem týká i některých modelů současných. Je jí omezená velikost vlastní aplikace. Úplným minimem, s nímž jsem se setkal, bylo 32kB, ale častěji se setkáme s omezením na 64kB, 80kB nebo 128kB. Tím samozřejmě vznikají další nepříjemnosti pro programátora, který musí buď omezit funkce, a snížit tak velikost výkonného kódu aplikace, nebo se musí vzdát některých zdrojů, grafiky, apod. Obvyklý postup je přijmout omezení v obou zmíněných kategoriích.

Na velikosti aplikace lze ušetřit i redukcí objektového modelu. Čím méně obsahuje aplikace tříd, tím menší je. To platí samozřejmě i pro funkce. Těm z vás, kteří jste obeznámeni se strukturou mezikódu Javy, je zřejmé, že lze dosáhnout vynecháním jedné funkce úspory řádově desítky

až stovky bytů jen na její prázdné definici. Dalších úspor lze dosáhnout například uchováváním hodnot v polích místo ve třídách.

Mobilní aplikace pro svoji tvorbu podporují jeden z dnes nejběžněji používaných jazyků, který je v podstatě celý založen na objektově orientovaném přístupu, ale současně svými omezeními brání jeho smysluplnému využití.

V případě knihovny je pak naprostou nutností její minimální velikost. Častěji je výhodnějším řešením vzdát se některých funkcí, které mají spíše usnadnit práci programátorovi, na úkor celkové velikosti kódu.

Z předchozích odstavců je zřejmé, že právě starší modely jsou největším problémem, který brání postupu technického zpracování mobilních aplikací. Komerčně úspěšný produkt musí tyto staré telefony podporovat, a přitom zde vývojáři aplikací narážejí na největší problémy. Kvůli výkonu je potřeba optimalizovat, ale na optimalizace není dostatečná paměť. V současné době se tento problém řeší hlavně vytvořením několika odlišných verzí téže aplikace a verze určené pro zastaralá zařízení bývají o poznání hůře zpracovány. Obvykle chybí grafické prvky a hra není zdaleka tak rozmanitá. Častá je i absence celých částí nebo funkcí.

6.4 Podpora J2ME

Knihovny pro tvorbu karetních her se to přímo netýká, ale pro úplnost si uveďme, že jedním z velkých úskalí je také samotná podpora jak konfigurací a profilů, tak zejména volitelných balíčků. Předpokládáme-li, že se omezíme pouze na standardní prvky jazyka a na několik abstraktních datových struktur lze s jistotou tvrdit, že nenarazíme na výrazný problém daný odlišnou implementací.

Pokud bychom využívali například možností MMAPI (MultiMedia API), narazíme na velké problémy s rozdílnou podporou na jednotlivých mobilních telefonech. Přitom právě tento volitelný balíček je prakticky jediným existujícím řešením pro zvukové efekty nebo hudební doprovod. Na problémy lze narazit například i u implementace datových proudů.

Prakticky jediným východiskem je otestovat aplikaci alespoň na jednom zástupci z podporované série mobilních telefonů. To sebou samozřejmě nese velmi vysoké nároky na finanční prostředky pro pořízení základního sortimentu testovacích telefonů.

Pokud bychom chtěli být schopni zajistit a otestovat funkci na dvou dnes nejběžnějších značkách telefonů, kterými jsou Nokia a Sony Ericsson a pouze na jejich nejrozšířenějších modelech, budeme potřebovat přibližně 15-20 testovacích telefonů.

Doufám, že se mi podařilo alespoň částečně nastínit fakt, že rozdíly v podpoře J2ME mohou mít obrovské časově-ekonomické důsledky.

Více se této kapitole věnovat nebudeme, protože knihovna, které je předmětem této práce, se této problematice vyhne. Jistě se však jedná o velmi rozsáhlé téma, které zahrnuje mnoho převážně negativních aspektů spojených s vývojem aplikací pro mobilní zařízení.

6.5 Obecné požadavky na mobilní aplikace

Do této kapitoly patří zejména reakce aplikace na vnější situace, persistence hry a rychlé pozastavení běhu. Jedná se o soubor doporučení, která se pro komerčně úspěšný produkt stávají nezbytnou nutností, a někteří z distributorů provádějí testy na vlastních zařízeních, aby ověřili, zda aplikace splňuje i tyto požadavky na „dobré chování“.

Při návrhu knihovny je potřeba dobře zvážit, jaký dopad na ni mají tyto obecné požadavky, a případně je nutné implementovat nezbytné funkce pro jejich řešení.

6.5.1 Příjem hovoru a SMS

Vzhledem k tomu, že aplikace běží na zařízení, které je primárně určeno k jinému účelu, nesmí aplikace znemožnit smysluplné užití zařízení v souladu s jeho bežným účelem.

Příjem SMS zprávy obecně nebývá problém, neboť dojde maximálně ke zvukové notifikaci. Bohužel i to může způsobit, že v aplikaci přestanou fungovat zvukové efekty, a proto je potřeba kontrolovat stav přehrávání zvuků. Na některých mobilních telefonech může dojít k minimalizaci aplikace, a řešení takového stavu je pak komplikovanější. Více si uvedeme v dalších odstavcích.

Při příjmu telefonního hovoru dojde zpravidla k minimalizaci aplikace. Naprostou nutností je reagovat na tuto situaci a hlavně okamžitě ukončit přehrávání zvukových efektů nebo hudby. Musíme také přepnout hru do stavu pauzy, protože hráč není během hovoru schopen reagovat, a proto bychom měli zajistit, že jeho rozehraná hra nedojde k újmě.

Obnovení aplikace z minimalizovaného stavu si většinou vyžádá nutnost překreslit celou herní scénu a znovu aktivovat zvuky. Obvykle se hra ponechá ve stavu pauzy, dokud hráč manuálně opět neaktivuje hru.

Častým problémem je ukončení a obnovení přehrávání zvukových efektů, kdy pak může například vzniknout potřeba odstranit všechny hudební přehrávače z paměti a znovu je vytvořit.

Problémem bývá i rozeznání příchozího hovoru. Existují celkem tři způsoby, jak to lze udělat. Standardně by měl telefon při externí události přepnout aplikaci do stavu pauzy a informovat o tom midlet prostřednictvím funkce *pauseApp* a funkce *startApp* při obnovení. Bohužel to není pravidlem. Jako řešení se pak nabízí funkce třídy *Canvas*, kterou je *hideNotify* volaná v okamžiku, kdy daná instance *Canvas* ztrácí svůj status zobrazované komponenty, což však kromě minimalizace aplikace nastane i při programové změně zobrazitelného prvku. Funkce *showNotify* je volána při zobrazení třídy *Canvas* a může sloužit k ukončení stavu pauzy.

Bohužel některé telefony nevolají ani jednu z těchto funkcí, a pak se jako řešení nabízí jediná možnost, kterou je měření času průchodu jednou herní smyčkou. Pokud tento čas přesáhne určitou povolenou hranici, dojde k pozastavení hry. Například některé telefony Samsung při minimalizaci vůbec neinformují aplikaci, ale pozastaví provádění, a tím daný čas smyčky naroste nad aktivační mez. Nedokážeme sice plně adekvátně a okamžitě zareagovat na příjem hovoru, ale pozastavení hry je i tak nutností. Hráč by mohl po ukončení hovoru ponechat ještě nějaký čas telefon bez dozoru a během této doby by nesmí hra za žádných okolností pokračovat v běhu.

Tím jsme se zmínili o problematice příjmu telefonního hovoru nebo SMS a nastínili jsme si očekávanou reakci na takovou situaci. Podobně jako rozdíly v podpoře J2ME i tato problematika vyžaduje testování na reálných zařízeních a jedná se o jedno z úskalí mobilních aplikací.

6.5.2 Perzistence hry

Mezi velmi ceněné a téměř vyžadované vlastnosti mobilních her a aplikací patří možnost uložit si aktuální stav hry. V případě mobilních her je tato nutnost dána zejména tím, že se jedná o zábavu, která má ukrátit například čekání na autobus, apod. Proto je vhodné mít možnost hru uložit, ukončit a později se k ní vrátit.

Ukládání a načítání dat z perzistentní paměti zajišťuje tzv. RMS (Record Management System). Jedná se o jednoduchou záznamově orientovanou databázi. Uložené záznamy jsou perzistentní a měly by odolat i úplnému vybití, případně dočasnému vyjmutí baterie.

Vzhledem k tomu, že je mobilní telefon vysoce personifikován a většinou k němu má přístup pouze jedna osoba, lze s výhodou ukládat pouze jeden stav hry. Pokud by to bylo možné, lze nabídnout i uložení více stavů.

Je potřeba si dát pozor na maximální velikost uložitelných dat. Tento problém se opět projevuje zejména u starších telefonů. Také bychom měli vzít v úvahu, že ukládání a načítání z RMS je jedním z nejpomalejších procesů v mobilních aplikací vůbec a že není vhodné provádět ho často.

Typické je hlavně ukládání na vyžádání a ukládání při ukončení aplikace.

6.5.3 Pauza

Už jsme se zmínili, že pauza je velmi důležitou při příjmu hovoru nebo SMS a při minimalizaci aplikace. Kromě této automatické pauzy je potřeba vytvořit mechanismus, který kdykoliv umožní hráči uvést hru do stavu pauzy manuálně. Obvykle se používá některá z kláves pod displejem, jenž jsou označovány jako *softkeys*.

V některých případech se stav pauzy řeší přechodem do menu. Uživatel by měl být vždy informován, že běh hry je pozastaven.

Pauza je jedním z klíčových aspektů a nesmíme ji opomenout.

6.6 Analýza požadavků

Z hlediska výstupu je potřeba flexibilní mechanismus, aby byl uživatel knihovny schopen vypořádat se s mnoha různými rozlišeními displejů a také sekundárními problémy jako je omezený počet barev, nízký kontrast a špatná čitelnost. Musíme poskytnout možnost pracovat s objekty velmi detailně, a přesto s určitým komfortem.

Vzhledem k rozdílným rozložením klávesnic a různým kódům jednotlivých kláves je potřeba zajistit převod kódů zařízení na virtuální kódy a ty zaslat knihovně. Ta poskytne možnost pro navigaci uvnitř balíčku, pro výběr karty a také pro navigaci mezi více balíčky.

V případě dotykových displejů je situace jednodušší. Knihovně pouze předáme souřadnice vzniku události. Musí být samozřejmě zajištěno uživatelem, že souřadnice zadané při vykreslování a souřadnice zaslané po vzniku události jsou ze stejného souřadného systému a nejsou afektované například translací.

Vzhledem k tomu, že účelem knihovny je řešit pouze část aplikace, musí výrazným způsobem šetřit systémové zdroje. Spotřeba paměti by měla být jen taková, aby knihovna pracovala dostatečně rychle a nedocházelo k opakovaným výpočtům.

Integrován by měl být mechanismus omezující časově náročné externí operace. V našem případě to platí zejména pro vykreslování, které mezi takové operace patří. Vykreslovány budou pouze nezbytně nutné objekty. Protože může, například z důvodu opětovného přepnutí do aplikace po přijetí telefonního hovoru, nastat potřeba překreslit celou scénu, a tudíž i všechny objekty, musí existovat možnost, jak explicitně požádat jednotlivé nebo i všechny objekty o jejich překreslení.

Implementujeme nezbytný základ tak, aby knihovna poskytovala všechny potřebné funkce, ale vzdáme se rozšíření, na jejichž použití nezávisí výsledný produkt. Upustíme od implementace dalších principů pro definici pravidel, apod.

Co se týče obecných požadavků na mobilní aplikace, nemusíme se o ně prakticky zajímat, protože jejich řešení je v kompetenci uživatele knihovny. Knihovny samotné se tyto požadavky netýkají.

7 Knihovna

Zanalyzovali jsme požadavky, které jsou na knihovnu kladeny, a můžeme zvolit nejvhodnější postup, jak knihovnu implementovat, abychom vyhověli všem nárokům. Neměli bychom ani zapomenout na případný budoucí rozvoj knihovny a navrhnout a vytvořit ji s dostatečně přehlednou koncepcí, která nebude klást meze expanzi.

7.1 Zvolené principy a dekompozice problému

Podívejme se, jakým způsobem můžeme rozdělit knihovnu na několik samostatných částí. Ty budou základem při návrhu objektového modelu a shrnují požadavky z předchozích kapitol.

7.1.1 Hra

Pro zajištění možnosti provádět operace nad všemi objekty a pro funkce, které budou ve hře poskytnuty pouze na jednom místě, vytvoříme objekt reprezentující samotnou hru. Uživateli knihovny bude poskytovat mechanismus pro vytváření karet a balíčků a sám se bude chovat jako balíček uchovávající karty. Oba typy objektů lze vytvořit samostatně a později je začlenit do hry, ale jsou-li vytváření prostřednictvím třídy reprezentující hru, budou začleněny automaticky.

U karet, a to i když vzniknou jako samostatný objekt a jsou vloženy do balíčku později, jsou jednotlivé instance evidovány a lze s nimi velmi jednoduše pracovat. Vždy máme k dispozici všechny karty, které jsou součástí hry.

Pokud nebudeme vytvářet karty samostatně, ale právě prostřednictvím mechanismu, který objekt hry nabízí, budou tyto karty vytvořeny a vloženy přímo do hry. Ta je, jak bylo zmíněno výše, také balíčkem. Odtud je lze pak pomocí stejných operací, jaké jsou dostupné i u všech ostatních balíčků, přesouvat.

Podobně jako karty lze vytvářet prostřednictvím objektu hry i balíčky. Vybrat si lze ze tří zmíněných druhů. Lze vytvořit balíček pro prosté uchování karet, zobrazitelný balíček nebo balíček interaktivní. Ihned po vytvoření je pak reference na balíček uchována ve hře. To je důležité zejména pro vykreslování, kdy dochází k postupnému překreslení všech herních objektů.

Jelikož mohou balíčky vznikat samostatně, nabízí herní objekt také možnost přidat je nebo je naopak odebrat z interní evidence. To je vhodné nejen pro separovaně vytvořené balíčky, ale například také pro odstranění balíčku, který reprezentuje hráče, jenž zrovna ukončil hru.

7.1.2 Karty

Každá karta ve hře bude reprezentována instancí velmi jednoduchého objektu, jehož primární funkcí bude zejména uchování informací o kartě. Z předchozích kapitol vyplývá, že potřebujeme znát barvu

a hodnotu karty, a rovněž je potřeba uchovat informace o tom, zda je karta viditelná (je vidět líc nebo rub) a zda je karta aktivní, nebo zvýrazněná.

Pro potřeby vykreslování bude objekt karty nést také informace o její pozici a velikosti. Jejich nastavení bude prováděno objektem balíčku tak, aby byla karta správně vykreslena jako jeho obsah.

Pro řazení karet v rámci balíčku poskytne každá karta funkci pro výpočet unikátního indexu podle zadaných parametrů. Jako primární klíč řazení lze zvolit barvu nebo hodnotu. Za sekundární klíč bude automaticky považován nepoužitý parametr. Důležitou volbou při řazení je také směr a lze zvolit, zda řadit vzestupně nebo sestupně.

7.1.3 Balíčky

Všechny typy balíčků mají společné operace pro práci s kartami. To zahrnuje zejména možnost karty získat, přidávat, odebírat, řadit a přesouvat. Ve skutečnosti bychom těmito funkcemi pokryli všechny požadavky, ale nezajistili bychom dostatečný uživatelský komfort, proto umožníme snadno získat kartu s vrchu, spodu a kdekoliv z prostředku balíčku. Také umožníme dávkový přesun karet, a to opět buď z vrchu, ze středu, a nebo přesun náhodných karet. Užitečné bude poskytnout i mechanismus na změnu stavu zvýraznění pro všechny karty, které balíček obsahuje. V neposlední řadě je velice užitečnou funkcí možnost zjistit počet karet v balíčku.

Zobrazitelný balíček musí disponovat funkcí pro jeho vykreslení. Tu lze použít samostatně pro vykreslení konkrétního balíčku. Objekt hry volá při překreslení scény zároveň vykreslení všech balíčků. Chceme-li balíček vykreslovat, musí existovat mechanismus pro nastavení parametrů, které s vykreslením souvisejí. Mezi parametry ovlivňující vzhled budou patřit především tyto:

- pozice balíčku a orientace vůči pozici (anchor point)
- odstup okraje balíčku od obsahu
- posun překrývajících se karet v horizontálním nebo vertikálním směru
- rozměry navigačních šipek a nastavení podmínek jejich zobrazení
- typ balíčku a viditelnost
- maximální a variabilní počet zobrazitelných karet.

Podívejme se podrobněji na vybrané z uvedených parametrů. Orientace vůči pozici udává, kde se nachází poziční bod balíčku. Typicky to bývá horní levý roh, ale pomocí kombinace hodnot pro vertikální (horní, dolní nebo střed) a horizontální (levý, pravý nebo střed) nastavení ho lze posunout prakticky kamkoliv. Uživatel knihovny je schopen objekty precizně pozicovat.

Maximálním a variabilním počtem zobrazitelných karet jsou myšleny dva různé parametry, z nichž v platnosti může být vždy pouze jeden. Oba parametry ovlivňují, kolik karet bude zobrazeno při vykreslování balíčku. V případě, že jich obsahuje více, je zobrazen pouze tento definovaný počet karet a zbytek zobrazen není. Rozdíl se projeví v okamžiku, kdy je karet v balíčku méně, než je hodnota parametru. V případě maximálního počtu zobrazitelných karet se velikost balíčku nikdy

nemění a po chybějících kartách zůstane prázdné místo. Variabilní počet zobrazitelných karet naopak zmenšuje velikost balíčku podle počtu obsažených karet.

Interaktivní balíček disponuje vlastnostmi a parametry zobrazitelného balíčku a přidává možnost výběru karty. Vzniká potřeba vykreslovat také navigační šipky, která u zobrazitelného balíčku neexistuje. Balíček musí umět zpracovat příjem události od klávesnice a dotykového displeje. Z požadavků předchozích kapitol je zřejmé, že musíme přidat také možnost pro nastavení aktivovatelnosti.

Podporu dotykového displeje zajistíme pomocí klikatelných oblastí, což je jeden ze způsobů, jak je možné příjem událostí od tohoto vstupního zařízení realizovat. Není potřeba složitých výpočtů pro určení cíle události, ale nelze ho použít před překreslením scény, neboť princip je založen na vytvoření oblasti a nastavení jejích rozměrů právě při vykreslování, kdy jsou nám známy. Jelikož se karty mohou překrývat, vzniká nutnost implementovat i mechanismus pro určení vrstvy, v níž se oblast nachází. Pro účely klikatelných oblastí rozšíříme interaktivní balíček o funkce, které budou zajišťovat reakci na aktivaci klikatelné oblasti.

S výhodou lze podporu dotykového displeje použít i pro reakci na událost od myši nebo jiného polohovacího zařízení, pokud využijeme knihovnu ve standardní edici Javy.

7.1.4 Rozhraní

Aby mohla knihovna komunikovat se svým okolím a zasílat požadavky na provedení nezbytně nutných operací, které z důvodů, o nichž jsme se zmínili v předchozích kapitolách, nebude sama implementovat, bude požadovat pro svoji smysluplnou práci implementované rozhraní.

Budeme potřebovat funkci pro vykreslení pozadí hry. Tu bude používat herní objekt v případě požadavku na překreslení celé herní scény. Také bude potřeba vykreslovat balíčky, navigační šipky a karty. Vykreslení karet se dále dělí podle jejich viditelnosti.

Kromě funkcí, které řeší operace neimplementované knihovnou, je nutné zajistit interakci knihovny s okolím. Potřebujeme předávat informace o tom, že byla prostřednictvím interaktivního balíčku vybrána karta, a velmi užitečnou funkcí bude notifikace o změně aktivního balíčku.

7.2 Objektový model

Knihovna bude složena z několika samostatných objektů a rozhraní a ze struktury objektů založených na dědičnosti.

Samostatnými třídami bude *Card*, která reprezentuje kartu, a *ClickableArea* sloužící pro zpracování událostí od dotykového displeje principem uvedeným v předchozích kapitolách.

Knihovna obsahuje dvě rozhraní. *GameListener* jako základní rozhraní mezi hrou a aplikací byl detailně popsán v předchozích kapitolách a očekávané funkce jsou, jak již bylo zmíněno výše, zaměřeny převážně na zasílání požadavků na vykreslení a zasílání informací o stavu hry.

SavableObject slouží zejména pro interní potřeby knihovny a definuje objekt implementující rozhraní jako uložitelný. Více o ukládání stavu hry bude uvedeno v další kapitole.

Balíček sloužící pro uchování karet je nejzákladnějším objektem a definuje funkce, které jsou společné všem balíčkům i hře. Nazveme ho *StoragePackage* a bude stát nejvýše ve struktuře objektového modelu balíčků, nebude mít žádného předka. Celý jeden zmiňovaný herní objekt je takto vyřešen.

Přestože hra není doslova vykreslitelným objektem, zajišťuje vykreslování dalších objektů a platí pro ni stejná pravidla pro omezení vykreslování, jež není nezbytně nutné. Vytvoříme třídu *DrawablePackage* jako potomka *StoragePackage*. Ta poskytne svým potomkům mechanismus pro řízení překreslení.

Objekt hry *Game* vytvoříme jako potomka *DrawablePackage*. Zděděním získáme všechny potřebné vlastnosti, které od tohoto objektu očekáváme jako od balíčku, a přidáme všechny funkce, jež vyplývají ze všech uvedených požadavků.

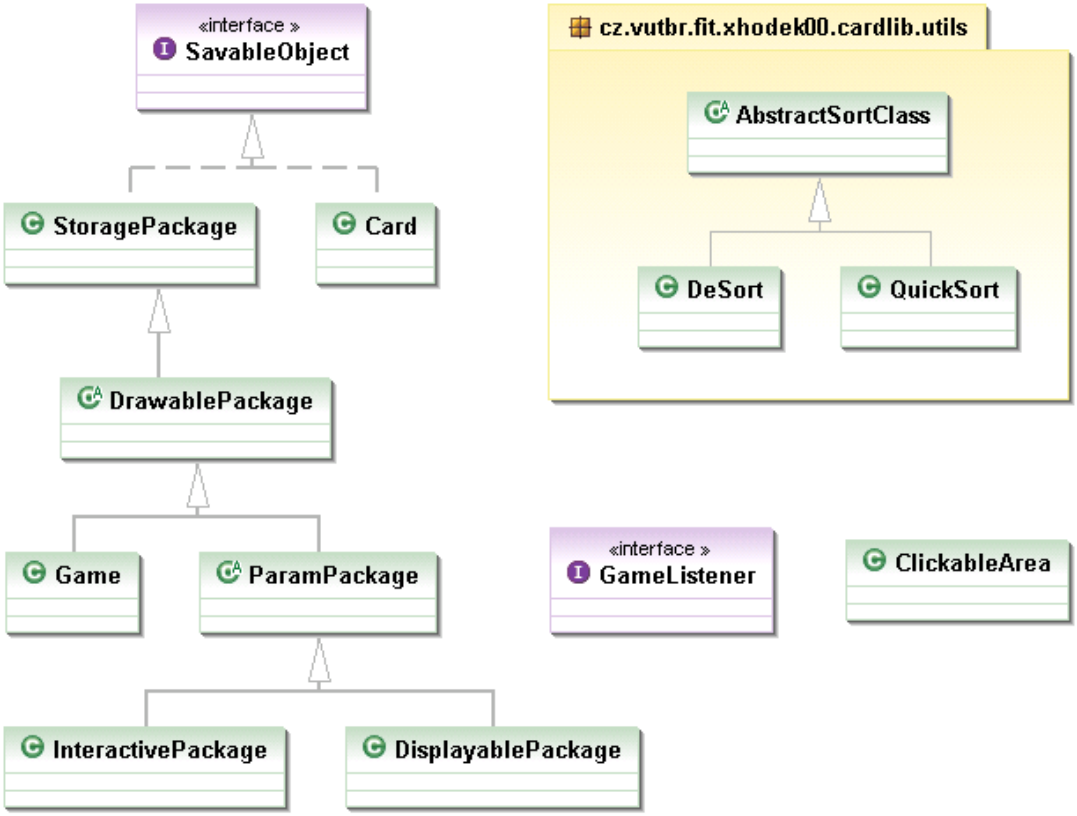
DrawablePackage poskytuje mechanismus pro řízení překreslení, ale objekty, které lze vykreslit, je potřeba rozšířit o zadání parametrů, o kterých jsme se zmiňovali v kapitolách výše. Tím dosáhneme vytvoření potomka *ParamPackage,* implementujícího systém pro uchovávání celočíselných parametrů. Ty lze nastavovat samostatně nebo prostřednictvím funkcí shlukujících společné parametry.

DrawablePackage je předkem pro *InteractivePackage* (interaktivní balíček) a *DisplayablePackage* (zobrazitelný balíček). U obou tříd jsou implementovány funkce pro vlastní vykreslení, a v případě interaktivního balíčku také funkce pro příjem událostí a nastavení, zda je balíček aktivovatelný či nikoliv.

Ještě jsme se nezmínili o abstraktní třídě *AbstractSortClass* a jejích potomcích *QuickSort* a *DeSort*. Abstraktní třída předepisuje, jak má vypadat implementace řadícího algoritmu pro potřeby knihovny. Řazení probíhá nad polem celých čísel a spolu s řazením musí dojít k přesunu položek pole objektů. Obvyklým užitím bude řazení indexů karet a polem objektů budou karty samotné.

Součástí knihovny jsou dvě implementace. Třída *QuickSort* řadí pole užitím stejnojmenného algoritmu podle velikosti hodnot prvního pole. Třída *DeSort* slouží k opačnému účelu. Neprovádí řazení, ale právě naopak pole náhodně přeskupí. V karetních hrách je tzv. míchaní balíčku velmi důležité.

Existence abstraktní třídy poskytuje možnost definovat nové řadící algoritmy. To může být velmi užitečné pro karetní hry, pro které nelze použít algoritmy, jež jsou součástí knihovny. Jedná se o část, kde je přehledná koncepce výhodným postupem do budoucna.



Obrázek 7.2.1: Objektový model knihovny (UML 2.1)

7.3 Ukládání stavu

Velmi důležitou vlastností, kterou musí knihovna disponovat je ukládání a načítání aktuálního stavu hry. Pro tento účel existuje rozhraní *SavableObject*. Každý objekt implementující toho rozhraní je možné uložit nebo načíst z datového proudu. S výhodou lze využít prezentovaného objektového modelu a ukládat pro balíčky v objektové struktuře pouze přidané vlastnosti a uložení či načtení zbytku parametrů provést voláním téže funkce předka.

Rozhraní *SavableObject* implementuje i samotný objekt hry a tímto způsobem lze uložit a opětovně načíst stav celé hry.

Přestože se při ukládání uloží všechny objekty a po načtení znovu vytvoří, dostáváme se do komplikované situace, neboť zaniknou platné reference. Pokud si v aplikaci udržujeme reference na balíčky, musíme po načtení hry tyto reference obnovit. Za tímto účelem umožňuje knihovna přiřadit vybraným balíčkům číselný identifikátor, podle kterého lze po načtení uložené hry opět získat referenci na ně.

Objekt hry uchovává reference na všechny balíčky, které hra obsahuje, a ty mohou být různého typu. Při ukládání herní objektu uložíme vlastnosti hry jako balíčku a poté cyklem projdeme všechny obsažené balíčky a uložíme. Ještě před vlastní data balíčku vložíme název jeho třídy jako textový řetězec. Při načítání budeme vytvářet třídy dynamicky pomocí *Class.forName*.

Uvedený postup umožňuje ukládat stejným způsobem i případné uživatelsky definované balíčky a je velmi výhodný. Problém může způsobit ukládání obfuskovaných her, kde jsou jména třídy změněna. Vše bude samozřejmě fungovat, dokud se případný soubor s uloženou hrou nepokusíme přenést do jiné verze aplikace, kde mohla obfuskace změnit jména jiným způsobem.

7.4 Realizace

Podívejme se na postup realizace knihovny. Jako vývojové prostředí jsem zvolil Eclipse a protože, jak jsme se zmínili v předchozích kapitolách, je potřeba zkompilované třídy preverifikovat, provádím spuštění hry pomocí Ant skriptu. Ten provede preverifikaci zkompilovaných souborů a přidá k nim grafické soubory a další zdroje a poté zajistí spuštění emulátoru. O kompilaci se stará automaticky Eclipse, jenž překompiluje soubory při každé změně zdrojových kódů.

Ze znalosti požadavků můžeme nejdříve vytvořit všechny samostatné objekty a rozhraní. Poté vytvoříme abstraktní třídu pro řazení a její dvě implementace. To je nutné, abychom byli schopni realizovat jakýkoliv balíček. Dále postupujeme podle objektového modelu směrem dolů od *StoragePackage* k jeho potomkům.

Zdrojové kódy knihovny jsem vygeneroval z vytvořeného UML Class diagramu a doplnil implementaci jednotlivých funkcí. UML diagram obsahoval pouze veřejné funkce, ale tam, kde to bylo výhodné, byly vytvořeny funkce privátní a pro případnou spolupráci objektů funkce chráněné (protected).

Pro testování knihovny jsem implementoval funkce *GameListeneru* nad rozhraním Swing a vytvořil jednoduchou testovací aplikaci. Vytvořené funkce jsem později poupravil, aby byly kompatibilní s micro edicí Javy a použil je při tvorbě demonstrační hry.

Více informací o realizaci hry poskytne stručná programová dokumentace, která tvoří přílohu této práce, nebo podrobná dokumentace (Javadoc), jenž se nachází na přiloženém datovém nosiči a jenž obsahuje popis všech veřejných, chráněných i privátních proměnných, konstant a funkcí.

7.5 Ukázka použití knihovny

Ukažme si jednoduchý postup, jak lze knihovnu použít. Všechny uvedené úseky kódu jsou ilustrativní a slouží pouze k vysvětlení principu. Předpokládejme, že se jedná o jednoduchou hru pro dva hráče. Každý z nich obdrží do balíčku 16 karet a každý z nich odhazuje karty do svého druhého balíčku. V odhazování se střídají. Ten, kdo odhodil kartu s nejvyšší hodnotou, získává bod. Pokud odhodí oba hráči kartu se stejnou hodnotou, pak nedostane bod ani jeden z nich. Hra končí, když hráčům nezůstanou v rukou žádné karty.

Pro jednoduchost vynecháme vykreslování, nastavování parametrů balíčků, apod. Nechci zde uvádět všechny možnosti knihovny, ale pouze demonstrovat základ práce s balíčky a kartami,

aby bylo vidět, jak navržený model knihovny funguje a jakým způsobem usnadňuje základní operace karetních her.

Nejdříve je potřeba implementovat rozhraní *GameListener*. Poté můžeme vytvořit instanci herního objektu *Game*.

```
// Vytvorime instanci hry.
Game game = new Game(gameListener, cardWidth, cardHeight);
```

Máme-li vytvořenou instanci hry, vložíme do ní karty. V tomto konkrétním případě vytvoříme 32-listové německé karty.

```
// vlozime karty A, 7, 8, 9, 10, J, Q, K
int[][] ranks = new int[][] { { 0, 7, 8, 9, 10, 11, 12, 13 } };
// pro barvy srdce, piky, kary a krize
int[] suits = new int[] { 0, 1, 2, 3 };
// vytvoreni karet
game.createCards(suits, ranks);
```

Nyní můžeme vytvořit balíčky, které se ve hře budou vyskytovat.

```
// balicky vytvorime jako pole
InteractivePackage[] rukaHrac = new InteractivePackage[2];
DisplayablePackage[] odhozHrac = new DisplayablePackage[2];

// balicky pro hrace 1
rukaHrac[0] = game.createInteractivePackage(DRAWTYPE_VERTICAL);
odhozHrac[0] = game.createDisplayablePackage(DRAWTYPE_ALL);

// balicky pro hrace 2
rukaHrac[1] = game.createInteractivePackage(DRAWTYPE_VERTICAL);
odhozHrac[1] = game.createDisplayablePackage(DRAWTYPE_ALL);
```

Vložíme každému z hráčů do ruky 16 karet.

```
// Vlozeni karet
game.moveRandomCards(16, rukaHrac[0]);
game.moveRandomCards(16, rukaHrac[1]);
```

Implementujeme funkci *cardSelected* z *GameListeneru*. Pro danou hru to lze provést velmi jednoduše takto:

```
// aktualni hrac
int aHrac = 0;

// body jednotlivych hracu
int[] score = new int[2];

public void cardSelected(Card card) {
  // presuneme kartu do odhazovaciho balicku
  rukaHrac[aHrac].moveCard(card, odhozHrac[aHrac]);

  // pokud jsme po tahu druheho hrace pricteme body
  if (aHrac == 1) {
    // zjistime jake karty jsou v obou baliccich
    int card1 = odhozHrac[0].getTopCard().getRank();
    int card2 = odhozHrac[1].getTopCard().getRank();
    // pricteme body vitezi
    if (card1 > card2) score[0]++;
    if (card2 > card1) score[1]++;
    // zkontrolujeme zda neni konec hry
    if (rukaHrac[aHrac].getCardCount() == 0) gameOver();
  }

  // prepneme na dalsiho hrace
  aHrac++;
  if (aHrac == 2) aHrac = 0;
}
```

Uvedená ukázka se týkala velice jednoduché hry a jednalo se pouze o výpis klíčových částí kódu. Přesto je vidět, že knihovna zásadním způsobem zjednoduší tvorbu takovéto hry. Kompletní zdrojový kód složitější demonstrační hry lze nalézt na přiloženém datovém nosiči. Popisem demonstrační hry se částečně zabývá následující kapitola.

8 Demonstrační hra

Jako demonstraci užití knihovny vytvoříme hru založenou na všeobecně známé karetní hře *Prší*. Pro lepší demonstraci knihovnou nabízených funkcí implementujeme několik různých variací hry. Zaměříme se pouze na velice stručný popis, neboť demonstrační hra není předmětem této práce. Zvláště pak popis implementace je velmi strohý a uvádí pouze základní vztahy mezi jednotlivými třídami. Pro detailní pochopení funkce doporučuji prostudovat přiložené zdrojové kódy hry.

8.1 Pravidla hry *Prší*

Hra *Prší* se hraje typicky v počtu 2 - 4 hráčů. Každý z hráčů obdrží na začátku hry 4 nebo 5 karet podle typu hry. Pro rychlou hru to bývá 8 karet. Zbytek karet tvoří talón, ze kterého si hráči berou karty v případě, že nemohou žádnou kartu odhodit nebo pokud musí brát. První karta z talónu se otočí a položí lícem nahoru. Tato karta tvoří základ balíčku pro odhazování karet.

Hráč, který je na tahu, může odhodit do odhazovacího balíčku kartu, která má buď stejnou barvu nebo hodnotu jako karta na vrchu odhazovacího balíčku. Výjimkou je pouze *kluk*, kterého lze vyhodit kdykoliv a po jeho odhození lze ohlásit změnu barvy na libovolnou jinou. Ve variaci s *žolíkem* pak lze odhodit *žolíka* na jakoukoliv kartu a také na něj lze jakoukoliv kartu odhodit.

Kromě *kluka* má speciální význam *eso*. Po jeho odhození musí další hráč odhodit buď také *eso*, nebo se musí vzdát svého tahu.

Speciální význam má kromě *esa* ještě *sedma* a v některých variantách také *piková dáma* nebo *žolík*. Jsou to tzv. karty, na které se bere. Typicky jsou to 2 karty na *sedmu*, 5 karet na *pikovou dámu* a 10 karet na *žolíka*. Existují i varianty, kdy se berou 4 karty na *srdcovou sedmu*. Při odhození karty, na kterou se bere, musí následující hráč buď brát nebo odhodit jinou kartu, na kterou se bere, a přesunout tím nutnost brát na dalšího hráče. V takovém případě musí následující hráč brát součet karet braných na jednotlivé po sobě vyhozené karty.

Hra končí, pokud se jeden z hráčů zbaví všech svých karet. Ostatní hráči pak dostávají podle hodnot karet, které jim zůstaly, trestné body. V jiné variantě hra končí až v případě, že zůstane poslední hráč a pořadí hráčů je dáno jejich postupným opouštěním hry.

Typicky se hraje s 32-listovými německými kartami, ale lze hrát i s jedním nebo dvěma balíčky 52-listových francouzských karet, případně i se *žolíky*.

Poslední variací pravidel, o které jsme se ještě nezmínili, je tzv. rychlá hra. V takové hře lze v jednom tahu odhodit více karet stejné hodnoty. Odhození první karty se řídí naprosto stejnými pravidly, jaká byla uvedena výše. Dále lze odhodit pouze karty se stejnou hodnotou. Lze odhodit i více *es* a karet, na které se bere. V takovém případě musí následující hráč vzít součet daný všemi odhozenými kartami. [15]

8.2 Popis implementace hry

Základem hry je midlet, který po spuštění zobrazí menu, prostřednictvím něhož lze pokračovat v rozehrané hře, existuje-li nějaká, vytvořit novou hru, dozvědět se informace o ovládání, autorovi a pravidlech hry, nebo hru ukončit. Celé menu je vytvořeno pomocí standardních komponent J2ME z balíčku *javax.microedition.lcdui*, aby byla zajištěna bezproblémová funkce na všech typech mobilních telefonů a aby bylo ovládání aplikace co nejbližší typickému chování daného zařízení. Jedná se o jednoduchou práci s formuláři a z hlediska implementace zde není nic komplikovaného ani zajímavého.

Vlastní hra *Prší* je implementována v samostatném balíku a je složena ze tříd *SevenGame* a *SevenAI* a z rozhraní *SevenGameListener*. Třída *SevenGame* zajišťuje celou funkci hry. Obsahuje instanci *Game* z knihovny pro tvorbu karetních her a provádí skrze ní všechny manipulace s kartami. Navenek poskytuje několik funkcí, které jsou potřeba pro zajištění základních operací ze strany kódu, jenž bude třídu využívat. Jedná se především o funkce pro změnu barvy, ukončení tahu, vstup od klávesnice či dotykového displeje, braní karet, atd.

Prostřednictvím rozhraní *SevenGameListener* žádá třída *SevenGame* o potřebné údaje nebo o provedení určité operace, případně notifikuje vnější kód o průběhu hry. Mezi funkce rozhraní patří funkce pro zjištění polohy balíčků, zobrazení intermezza, notifikaci o stání hráče či povinnosti dobrat karty, ukončení hry, funkce pro požadavek na zobrazení dialogu pro změnu barvy, apod.

Poslední třídou pro hru *Prší* je *SevenAI*, která implementuje jednoduchou umělou inteligenci, pomocí níž lze nahradit jednoho nebo více hráčů. Samotný princip umělé inteligence je detailněji rozebrán v následující kapitole.

Ke své funkci potřebuje třída *SevenGame* nutně implementaci rozhraní *SevenGameListener* i *GameListener* knihovny.

Vlastní hra je pak realizována pomocí třídy odvozené od *Canvas*, která navíc implementuje *GameListener* a zároveň i *SevenGameListener*. Třída zajišťuje vykreslování karet, balíčků, atd. Rovněž reaguje na požadavky od obou rozhraní. Události od klávesnice nebo dotykového displeje přeposílá *SevenGame*. Jedinou skutečně samostatnou funkcí této třídy je opakované vykreslování hry ve vlastním vlákně a reakce na některé vnější události.

I z tohoto stručného popisu implementace je vidět, jak slabě je knihovna závislá na konkrétním implementačním prostředí, jsou-li splněny všechny podmínky, a že lze tedy velmi snadno vytvořit aplikaci podporující široké spektrum mobilních zařízení. Zvláště výhodná je zde minimální závislost na vykreslování.

8.3 Umělá inteligence

Umělá inteligence je v demonstrační hře vytvořena velice jednoduše. Pro aktuálního hráče je provedena analýza stavu hry. Projdou se všechny karty, které má v daném okamžiku ve svém balíčku, a hledají se ty, které je možno odhodit. Jsou zahrnuty i stavy, kdy musí aktuální hráč brát karty, nebo kdy má stát.

Každá z karet, která vyhovuje pro odhození, je ohodnocena podle svého významu. Nejdříve se snaží umělá inteligence zbavit běžných karet, poté karet, na které se bere, a úplně nakonec si nechává *kluky*, které je možné odhodit téměř vždy, a jsou tedy k ukončení hry zdaleka nejvýhodnější.

Pokud se umělá inteligence dostane do situace, kdy nemůže odhodit žádnou kartu, pak si dobere. Pokud by měla v daném okamžiku k dispozici *kluka*, odhodí ho a změní na barvu, od které má nejvíce karet.

V režimu rychlé hry se provádí odhazování karet opakovaně. První odhození se řídí stejnými pravidly jako v běžné hře, dále jsou odhazovány pouze karty stejné hodnoty. Není implementována pokročilá funkce odhazování s výběrem nejvhodnější barvy zakončení.

8.4 Testování

Demonstrační hru jsem otestoval na všech telefonech, které jsem měl k dispozici. Testoval jsem zejména následující faktory:

- přizpůsobení rozlišení displeje daného zařízení, dobrá čitelnost objektů
- okamžitá reakce na vstupy uživatele, minimální doba zpracování jednotlivých operací
- uložení při ukončení a načtení při opětovném spuštění
- dostupnost všech funkcí pomocí klávesnice nebo dotykového displeje
- bezchybný běh pro různá nastavení, zejména pro ta, která vyžadují nejvíce paměti

Aby hra splnila všechny požadavky, musela vyhovovat všem uvedeným bodům. Nejdůležitější byla okamžitá reakce a bezchybný běh po celou dobu testování. Následuje tabulka jednotlivých telefonů, jejich stručný popis a komentář k testování. Uváděny jsou pouze rozdíly proti požadavkům.

Typ telefon	**Stručný popis zařízení**	**Poznámky k testování**
Nokia 3100	Telefon společnosti Nokia není vybaven otevřeným operačním systémem. Displej má rozlišení 128x128. Podporuje konfiguraci CLDC 1.0 a profil MIDP 1.0.	Testována byla verze pro MIDP 1.0 a verze využívající Nokia UI API pro zajištění celoobrazovkového režimu hry. Reakce hry jsou pomalejší. Tento telefon patří mezi

		nejpomalejší telefony, se kterými se lze setkat.
Nokia 6111	Telefon společnosti Nokia není vybaven otevřeným operačním systémem. Displej má rozlišení 128x160. Podporuje konfiguraci CLDC 1.1 a profil MIDP 2.0.	Všechny požadavky byly splněny.
Benq-Siemens S68	Telefon společnosti Benq-Siemens není vybaven otevřeným operačním systémem. Displej má rozlišení 132x176. Podporuje konfiguraci CLDC 1.1 a profil MIDP 2.0.	Všechny požadavky byly splněny.
Nokia 6600	Telefon společnosti Nokia je vybaven operačním systémem Symbian. Displej má rozlišení 176x208. Podporuje konfiguraci CLDC 1.0 a profil MIDP 2.0.	Všechny požadavky byly splněny.
Sony Ericsson D750i	Telefon společnosti Sony Ericsson není vybaven otevřeným operačním systémem. Displej má rozlišení 176x220. Podporuje konfiguraci CLDC 1.1 a profil MIDP 2.0.	Všechny požadavky byly splněny. Displej ma malé rozměry, a proto byla zhoršená čitelnost objektů.
LG KG800	Telefon společnosti LG není vybaven otevřeným operačním systémem. Displej má rozlišení 176x220. Podporuje konfiguraci CLDC 1.1 a profil MIDP 2.0.	Všechny požadavky byly splněny.
Nokia 6300	Telefon společnosti Nokia není vybaven otevřeným operačním systémem. Displej má rozlišení 240x320. Podporuje konfiguraci CLDC 1.1 a profil MIDP 2.0.	Všechny požadavky byly splněny. Displej ma malé rozměry, a proto byla zhoršená čitelnost objektů.
Nokia 9300	Komunikátor společnosti Nokia je vybaven operačním systémem Symbian a displejem s rozlišením 640x200. Podporuje konfiguraci CLDC 1.0 a profil	Všechny požadavky byly splněny. Drobným nedostatkem bylo ovládání hry. Všechny funkce byly dostupné pomocí

	MIDP 2.0.	klávesnice, ale rozmístění ovládacích prvků bylo nevhodné. Vzhledem k rozlišení byly karty příliš široké.
Sony Ericsson P910i	Komunikátor společnosti Sony Ericsson je vybaven operačním systémem UIQ, dotykovým displejem s rozlišením 208x320 a podporuje konfiguraci CLDC 1.0 a profil MIDP 2.0.	Všechny požadavky byly splněny.

9 Závěr

Cílem této bakalářské práce bylo navržení a implementace knihovny pro tvorbu karetních her na mobilních zařízeních. Po důkladném seznámení se s požadavky karetních her, implementačním prostředím a případnými omezeními ze strany cílových zařízení, byl navržen optimální způsob implementace. Následně byla knihovna vytvořena a použita v demonstrační hře, aby se prokázalo, že nedošlo při návrhu k chybám, které by znemožnily její praktické použití.

S pomocí knihovny byla implementace herní logiky demonstrační hry bezproblémovou záležitostí. Jednotlivé potřebné operace bylo možno provádět velice rychle a přehledně a díky nástrojům, které knihovna poskytuje, byla snadno vytvořena hra, kterou lze ovládat klávesnicí nebo dotykovým displejem, v níž lze velice flexibilně měnit vzhled karet i ostatních částí a rovněž snadno přidávat další vlastnosti.

Hlavním cílem této bakalářské práce bylo vytvoření požadavků, na jejichž základě pak byla implementována velmi obecná knihovna pro tvorbu karetní her. Tento hlavní cíl byl splněn. Stále zde však zůstává prostor pro další rozvoj knihovny. Rozšířit by ji bylo možné hlavně v oblasti definice pravidel. K usnadnění práce s knihovnou by přispěla alespoň částečná implementace prerekvizitních podmínek a podobných přístupů sloužících ke zjednodušení zápisu ověřování pravidel.

Oblastí, ve které by se dala knihovna dále rozšiřovat, je také podpora internetového nebo bluetooth spojení pro transparentní přenos herních dat. Pro hru jednoho hráče by byla výhodná užší podpora počítačem řízených hráčů. Například varianta interaktivního balíčku, nereagující na vstupy uživatele, ale dotazující se vnějšího kódu umělé inteligence, by značně zjednodušila integraci hráčů s alternativní inteligencí.

Vytvořená demonstrační hra si našla velkou oblibu mezi lidmi, kteří mi ji pomáhali otestovat a mnoho z nich si ji ponechalo a aktivně ji používá ke zkrácení dlouhé chvíle. Doufám, že knihovna bude úspěšně využita i při realizaci dalších her ke spokojenosti jejich autorů i hráčů.

Literatura

[1] *Historie karet a karetních her* [online]. 2006 [cit. 2008-04-27]. Cs. Dostupný z WWW: <http://www.karetnihry.ic.cz/index.php?option=com_content&task=view&id=14>.

[2] *MCLEOD, John. Rules of Card Games : Alphabetical Index of Card Games [online]. c1995 , last updated 27th September 2007 [cit. 2008-04-27]. En. Dostupný z WWW: <http://www.pagat.com/alpha.html>.*

[3] Www.portal.czechgamer.com. *Magic The Gathering : Portal* [online]. c2005 [cit. 2008-04-20]. Cs. Dostupný z WWW: <http://www.portal.czechgamer.com/>.

[4] NEKULA, Jan. *Doomtrooper Arena* [online]. [2006] [cit. 2008-04-20]. Cs. Dostupný z WWW: <http://dtarena.com/>.

[5] *Wikiknihy : Pravidla karetních her/Taroky* [online]. 2007 , Stránka byla naposledy editována 29. 11. 2007 v 23:02. [cit. 2008-04-17]. Cs. Dostupný z WWW: <http://cs.wikibooks.org/wiki/Pravidla_karetních_her/Taroky>.

[6] Sun Microsystems. *Java ME at a Glance* [online]. c1994 [cit. 2008-04-15]. En. Dostupný z WWW: <http://java.sun.com/javame/index.jsp>.

[7] Sun Microsystems. *Connected, Limited Device Configuration* [online]. Specification version: 1.0a. 2000 [cit. 2008-04-14]. En. Dostupný z WWW: <http://jcp.org/aboutJava/communityprocess/final/jsr030/index.html>.

[8] Sun Microsystems. *Mobile Information Device Profile (JSR-37)* [online]. Version: 1.0a. 2000 [cit. 2008-04-14]. En. Dostupný z WWW: <http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html>.

[9] Sun Microsystems. *Connected Limited Device Configuration* [online]. Specification version: 1.1. 2003 [cit. 2008-04-14]. En. Dostupný z WWW: <http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html>.

[10] Sun Microsystems: *Mobile Information Device Profile for Java™ 2 Micro Edition* [online]. Version: 2.0. 2002 [cit. 2008-04-14]. En. Dostupný z WWW: <http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html>.

[11] ISMAIL, M.H. *Mobile Programming Pit Stop : What's new in JavaME MIDP 2.1 - Nokia S40 5th Edition* [online]. 2007 [cit. 2008-04-27]. En. Dostupný z WWW: <http://mobilepit.com/06/whats-new-in-javame-midp-21-nokia-s40-5th-edition.html>.

[12] GIGUERE, Eric. *J2ME Optional Packages* [online]. 2002 [cit. 2008-04-16]. En. Dostupný z WWW: <http://developers.sun.com/mobility/midp/articles/optional/>.

[13] Sun Microsystems. *JSRs: Java Specification Requests : List by JCP Technology* [online]. c1995 [cit. 2008-04-16]. En. Dostupný z WWW: <http://www.jcp.org/en/jsr/tech?listBy=1&listByType=platform>.

[14] Enough Software. *J2ME Polish: APIs : Device by APIs* [online]. [2004] [cit. 2008-04-20]. En. Dostupný z WWW: <http://www.enoughsoftware.com/devices/apis.html>.

[15] BAŤHA, Matěj. *Prší - Mau-mau - Pony* [online]. [1996] [cit. 2008-04-23]. Cs. Dostupný z WWW: <http://www.enoughsoftware.com/devices/apis.html>.

Seznam příloh

Příloha 1. Stručná programová dokumentace

Příloha 2. Popis demonstrační hry

Příloha 3. CD/DVD, obsahuje:

- stručný a podrobný objektový model
- zdrojové kódy knihovny
- zkompilovanou knihovnu ve formátu JAR
- programovou dokumentaci ke knihovně
- podrobnou programovou dokumentaci ke knihovně a demonstrační hře
- demostrační hru ve verzích MIDP 1.0, MIDP 2.0 a Nokia UI API
- zdrojové kódy demonstrační hry
- demonstrační hru umístěnou v emulátoru a spustitelnou na osobním počítači

1 Stručná programová dokumentace

Uveďme si vždy funkcionalitu, kterou od objektu očekáváme a která by měla být dostupná uživateli knihovny. Záměrně přitom vynecháme třídy a rozhraní *ClickableArea*, *SavableObject*, *DeSort* a *QuickSort,* a to buď z toho důvodu, že nenabízí žádnou funkcionalitu dostupnou uživateli knihovny, a nebo implementují jasně definované funkce svého předka či jejich funkce nemají význam bez konkrétní implementace.

Vynecháme privátní a chráněné funkce, které slouží buď objektu samotnému nebo ke komunikaci mezi objekty a z hlediska uživatele nejsou důležité. Nutnost vytvořit je vyplyne z implementace veřejných funkcí.

Kompletní seznam všech funkcí jednotlivých objektů včetně popisu a seznamu parametrů lze najít v nápovědě ke knihovně, která je na přiloženém datovém nosiči.

Rozhraní GameListener

- drawPackage() – vykreslení balíčku
- drawCard() – vykreslení karty lícem
- drawCardReverse() – vykreslení rubu karty
- drawBackground() – vykreslení pozadí hry
- drawArrow() – vykreslení navigační šipky
- cardSelected() – notifikace o výběru karty uživatelem
- packageActivated() – notifikace o změně aktivního balíčku

Třída Card

- konstruktor pro vytvoření karty s danou barvou a hodnotou
- konstruktor pro vytvoření karty z datového proudu
- getSuit() a getRank() – vrací barvu a hodnotu karty
- getX(), getY(), getWidth() a getHeight() – vrací umístění a rozměry karty
- isActive() – slouží ke zjištění, zda je daná karta aktivní
- setHighlighted() a isHighlighted() – slouží ke zjištění a nastavení zvýraznění karty
- setVisible() – nastavuje viditelnost karty, tj. zda je vidět rub nebo líc
- load() a save() – načítání a ukládaní parametrů karty do datového proudu

Třída AbstractSortClass

- sort() – zajistí seřazení pole objektů dle implementovaného algoritmu

Třída StoragePackage

- insertCard() – vloží kartu do balíčku
- getTopCard() – vrátí kartu z vrchu balíčku
- getBottomCard() – vrátí kartu ze spodu balíčku
- getCardAt() – vrátí kartu na specifikované pozici
- moveCard() – přesune vybranou kartu z tohoto balíčku do jiného
- moveCardsFromTop() – přesune zvolený počet karet z vrchu balíčku
- moveCardsFromBottom() – přesune zvolený počet karet ze spodu balíčku
- moveRandomCards() – přesune zvolený počet náhodně vybraných karet
- setAllHighlighted() – nastaví stav zvýraznění pro všechny obsažené karty
- sort() – seřadí karty dle zadaných kritérií
- getCardCount() – vrací počet karet v balíčku
- switchCards() – vymění mezi sebou karty v rámci tohoto balíčku
- setIdentifier() – nastaví číselný identifikátor balíčku
- isIdentifiedBy() – zjišťuje, zda je balíček identifikován daným identifikátorem
- load() a save() – načítání a ukládaní parametrů do datového proudu

Třída DrawablePackage

- forceRepaint() – nastaví příznak požadavku na překreslení objektu
- lockRepaint() – uzamkne příznak překreslení na zvolené hodnotě
- moveCard() – rozšíříme o automatické zajištění překreslení po provedení
- insertCard() – rozšíříme o automatické zajištění překreslení po provedení
- sort() – rozšíříme o automatické zajištění překreslení po provedení
- draw() – abstraktní funkce pro vykreslení, potomci ji musí implementovat
- load() a save() – načítání a ukládaní parametrů do datového proudu

Třída ParamPackage

- copyParamsFrom() – zkopíruje parametry z jiného objektu
- setParam() a getParam() – nastavuje a zjišťuje hodnotu vybraného parametru
- setType() – nastaví typ balíčku
- setPosition() – nastaví pozici balíčku
- setPadding() – nastaví odstup okraje od obsahu
- setConfiguration() – nastaví posun překrývajících se karet a maximální a variabilní počet zobrazitelných karet

- setArrows() – nastaví detaily vykreslování navigačních šipek
- setVisible() – nastavuje viditelnost obsažených karet
- getX(), getY(), getWidth() a getHeight() – vrací umístění a rozměry balíčku
- isActive() – zjišťuje, zda je daný balíček aktivní
- load() a save() – načítání a ukládaní parametrů do datového proudu

Třída Game

- konstruktor pro vytvoření nové hry a nastavení GameListeneru a rozměrů karet
- konstruktor pro vytvoření hry z datového proudu
- setCardSize() – změna rozměrů karet
- draw() – vykreslení celé herní scény
- setActivePackage() – nastaví zvolený balíček jako aktivní
- getActivePackage() – vrací aktivní balíček
- isPackagetActive() – zjišťuje, zda je balíček aktivní
- createStoragePackage() – vytvoří balíček pro uchování karet
- createDisplayablePackage() – vytvoří zobrazitelný balíček
- createInteractivePackage() – vytvoří interaktivní balíček
- addPackage() a removePackage() – umožňuje vkládat a odebírat balíčky ze hry
- createCard() – vytvoří kartu s danou barvou a hodnotou
- createCards() – umožňuje dávkové vytváření karet dle různých kritérií
- setAllHighlighted() – nastaví stav zvýraznění pro všechny karty ve hře
- getPackagetByIdent() – vrací balíček s daným identifikátorem
- performKeyEvent() – zajistí reakci na událost od klávesnice
- performPointerEvent() – zajistí reakci na událost od dotykového displeje
- load() a save() – načítání a ukládaní parametrů do datového proudu

Třída DisplayablePackage

- kontruktor pro vytvoření zobrazitelného balíčku určitého typu a náležejícího k určité hře
- draw() – vykreslí balíček

Třída InteractivePackage

- kontruktor pro vytvoření interaktivního balíčku určitého typu a náležejícího k určité hře
- setActivable() a isActivable() – nastavuje a zjišťuje, zda je daný balíček aktivovatelný

- draw() – vykreslí balíček
- isActive() – zjišťuje, zda je daný balíček aktivní
- load() a save() – načítání a ukládání parametrů do datového proudu

Konstanty

Knihovna definuje na různých místech několik konstant. Protože jsou poměrně důležité a umožňují lépe pochopit rozšiřitelnou koncepci vybraných částí a omezení platformní závislosti knihovny, podívejme se nyní na ně.

DisplayablePackage disponuje třemi konstantami pro ovlivnění viditelnosti obsažených karet. Jsou jimi:

- DRAWTYPE_ALL – všechny karty sdílí viditelnost balíčku
- DRAWTYPE_ONE – všechny karty sdílí viditelnost balíčku vyjma vrchní karty, jejíž viditelnost je opačná
- DRAWTYPE_CUSTOM – balíček neovlivňuje viditelnost karet a záleží na nastavení jednotlivých karet

U InteractivePackage vždy přebírají karty viditelnost balíčku, a proto nejsou předchozí konstanty nutné. Na rozdíl od *DisplayablePackage* však existují navigační šipky a je potřeba zvolit jejich orientaci. Z tohoto důvodu definujeme tyto konstanty:

- DRAWTYPE_HORIZONTAL – navigace probíhá vodorovně, šipky jsou zobrazeny vlevo a vpravo a aktivní karta se mění pomocí klávesy vlevo a vpravo.
- DRAWTYPE_VERTICAL – navigace probíhá vertikálně, šipky jsou zobrazeny nahoře a dole a aktivní karta se mění pomocí klávesy nahoru a dolů.

Předchozí balíčky mají společného předka, kterým je *ParamPackage* a který obsahuje následující konstanty pro jednotlivé parametry.

- PARAM_PADDING_LEFT – odstup okraje od obsahu vlevo
- PARAM_PADDING_RIGHT – odstup okraje od obsahu vpravo
- PARAM_PADDING_TOP – odstup okraje od obsahu nahoře
- PARAM_PADDING_BOTTOM – odstup okraje od obsahu dole
- PARAM_CARDMOVE_LEFT – posun překrývajících se balíčků na ose X
- PARAM_CARDMOVE_TOP – posun překrývajících se balíčků na ose Y
- PARAM_POSITION_X – pozice balíčku na ose X
- PARAM_POSITION_Y – pozice balíčku na ose Y
- PARAM_ANCHOR – orientace vůči pozici
- PARAM_TYPE – typ balíčku
- PARAM_VISIBLE – viditelnost balíčku
- PARAM_MAXCOUNT – maximální počet zobrazitelných karet

- PARAM_VARCOUNT – variabilní počet zobrazitelných karet
- PARAM_ARROW_WIDTH – pro horizontální balíček šířka, pro vertikální výška oblasti pro zobrazení navigační šipky
- PARAM_ARROW_STILLSHOWED – nastavení stálého zobrazování navigačních šipek

Všechny uvedené parametry se vztahovaly ke konkrétním objektům a dá se předpokládat, že nebudou využity při rozšiřování knihovny. Pokud by využity byly, bude to v potomcích daných tříd. Pro následující parametry platí obecnost, neboť pravděpodobnost jejich využití je podstatně vyšší nebo se jedná to konstanty, které souvisejí s více odlišnými objekty.

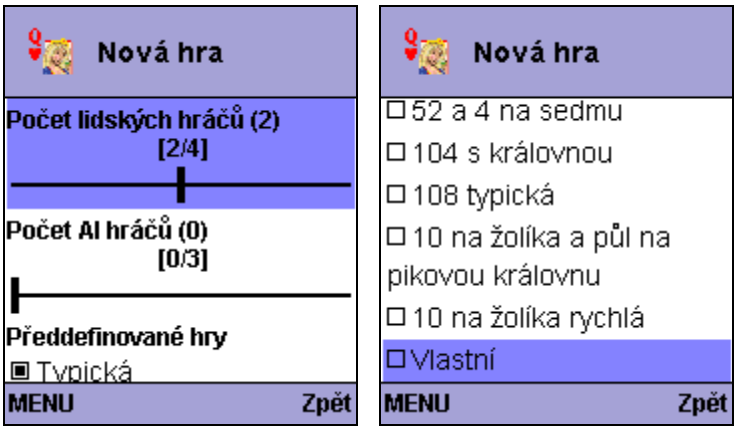
- SORT_BY_SUIT – řazení podle barvy
- SORT_BY_RANK – řazení podle hodnoty
- SORT_RANDOM – míchání karet
- SORT_ASC – vzestupné řazení
- SORT_DESC – sestupné řazení
- ARROW_LEFT – označuje navigační šipku doleva, například při vykreslování
- ARROW_RIGHT – označuje navigační šipku doprava, například při vykreslování
- ARROW_TOP – označuje navigační šipku nahoru, například při vykreslování
- ARROW_DOWN – označuje navigační šipku dolů, například při vykreslování
- ANCHOR_LEFT – poziční bod je vlevo
- ANCHOR_RIGHT – poziční bod je vpravo
- ANCHOR_HCENTER – poziční bod je na středu (horizontálně)
- ANCHOR_TOP – poziční bod je nahoře
- ANCHOR_BOTTOM – poziční bod je dole
- ANCHOR_VCENTER – poziční bod je na středu (vertikálně)
- KEY_NOKEY – žádná klávesa nebyla stisknuta
- KEY_LEFT – virtuální kód pro klávesu vlevo, navigace v balíčku
- KEY_RIGHT – virtuální kód pro klávesu vpravo, navigace v balíčku
- KEY_UP – virtuální kód pro klávesu nahoru, navigace v balíčku
- KEY_DOWN – virtuální kód pro klávesu dolů, navigace v balíčku
- KEY_CARD_SELECT – virtuální kód pro potvrzení výběru karty
- KEY_PACK_NEXT – virtuální kód pro další balíček, navigace mezi balíčky
- KEY_PACK_PREV – virtuální kód pro předchozí balíček, navigace mezi balíčky

2 Popis demonstrační hry

Při spuštění hry se zobrazí hlavní menu, prostřednictvím kterého lze pokračovat v rozehrané hře, spustit novou hru, zobrazit informace o autorovi, ovládání či pravidlech, a nebo hru ukončit. Pokračovat v rozehrané hře lze pouze pokud existuje uložená hra. V případě, že není žádná hra rozehrána bude hráč informován upozorňujícím dialogem.

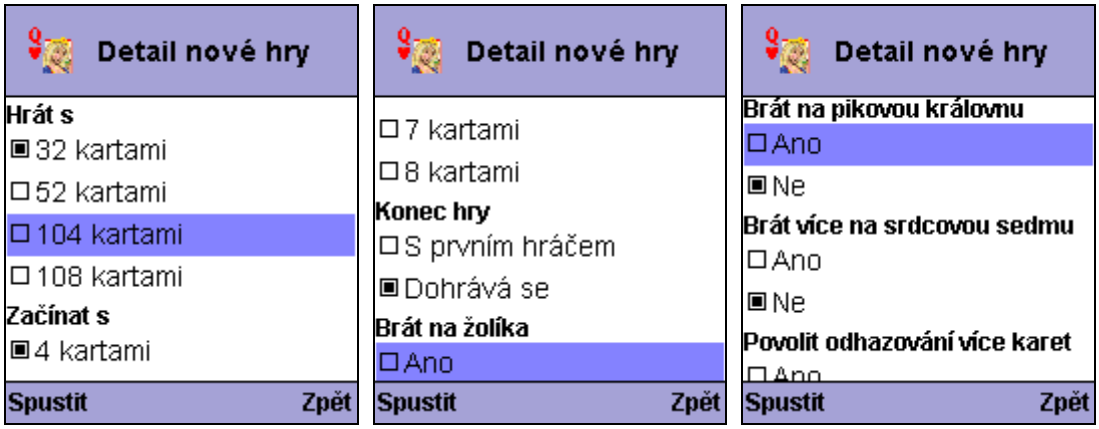
2.1 Hlavní menu hry

Zobrazení informací o autorovi, ovládání a nebo pravidlech popisovat nebudeme, protože se jedná o standardní formulář, jenž obsahuje příslušný text. Volba pokračování v rozehrané hře přepne do hry a ukončení hry ji ukončí. Zajímavá je volba „Nová hra“.



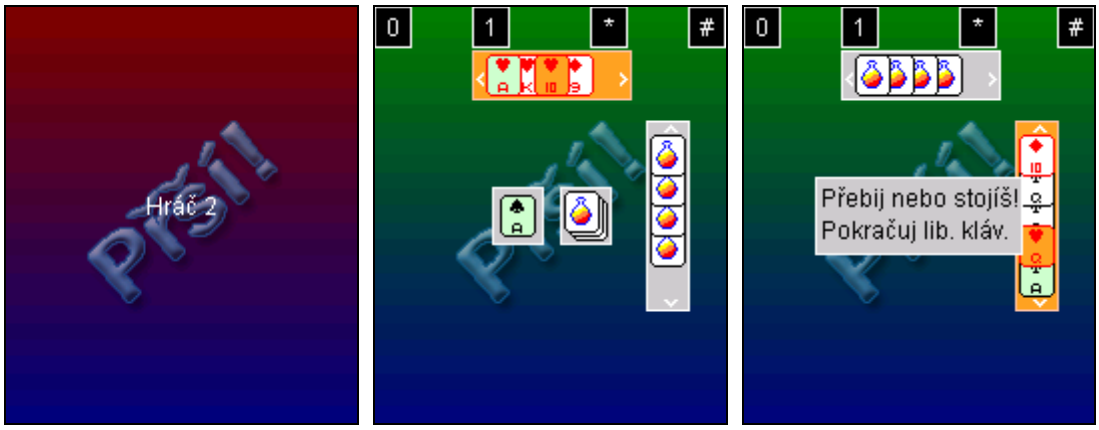
2.2 Nastavení nové hry

Menu nové hry umožňuje nastavit počet lidských a počítačem řízených hráčů a vybrat jeden z předdefinovaných typů hry. Z menu nové hry můžeme hru přímo spustit a nebo lze zobrazit detailní nastavení hry.



2.3 Detaily nastavení nové hry

Detailni nastavení umožňuje individuálně ovlivnit jednotlivé aspekty hry. Z menu detailního nastavení nové hry se lze vrátit zpět do menu nové hry a nebo spustit hru.



2.4 Vlastní hra

Na obrázcích výše jsou vidět tři základní stavy hry. První obrázek znázorňuje tzv. intermezzo. To zabraňuje hráči, který ukončil tah, aby viděl karty soupeře a naopak následující hráč nevidí karty předchozího.

Druhý obrázek zobrazuje vlastní hru. Prostřednictvím šipek se může hráč pohybovat ve svém balíčku. Oranžovým pozadím je zvýrazněna vybraná karta a zeleně jsou zvýrazněny karty, které lze v dané situaci odehrát. Balíčky ostatních hráčů mají obsažené karty zobrazeny rubem. Uprostřed hrací plochy je talón a odhazovací balíček. V horní části jsou tlačítka pro ovládání dotykovým displejem.

Na posledním obrázku je zachycena zpráva informující aktuálního hráče o důležitých událostech, k nimž došlo v tahu předchozích hráčů.