VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

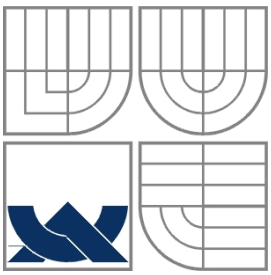
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PLATFORMA PRO REALIZACI DESKOVÝCH HER NA POČÍTAČI

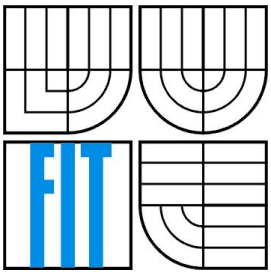
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ALEŠ KŘENEK
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PLATFORMA PRO REALIZACI DESKOVÝCH HER NA POČÍTAČI

FRAMEWORK FOR BOARD GAME REALIZATION ON COMPUTER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ALEŠ KŘENEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VÍTĚZSLAV BERAN, Ph. D.

BRNO 2013

Abstrakt

Práce si klade za cíl vytvořit interpret a uložení herních prvků. Jejím primárním cílem je vytvořit platformu pro tvorbu deskových her, která bude použitelná i člověkem, který nemá znalosti z oboru programování. Sekundárními cíli projektu je multiplatformní řešení, včetně možné komunikace dvou různých platforem, přenositelnost editorem vytvořených definic deskových her a uložení veškerých vytvořených dat v podobě čitelné člověkem.

Abstract

Work aims to create Board games interpreter and repository of game elements. The primary goal of work is to create a platform for board game realization, witch can be used also by someone who does not have knowledge in the field of programming. Secondary objectives of the project is a cross-platform solution, including possible communication between two different platforms, portability of created board game definitions and store of all data in human-readable form.

Klíčová slova

deskové hry, uživatelská přístupnost, multiplatformní řešení

Keywords

board games, user-friendliness, multiplatform solution

Citace

Aleš Křenek: Platforma pro realizaci deskových her na počítači, bakalářská práce, Brno, FIT VUT v Brně, 2013

Platforma pro realizaci deskových her na počítači

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vítězslava Berana, Ph.D.
Další informace mi poskytli Dr. Ing. Petr Peringer a Vladimír Chvátil.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Aleš Křenek
30.4.2013

Poděkování

Rád bych poděkoval především mému vedoucímu, za jeho ochotu poradit a za to že si vždy na mě našel čas když jsem to potřeboval.

Dále bych rád poděkoval Vladimíru Chvátilovi, jelikož snaha realizovat na PC jeho jedinečné deskové hry byla úvodní motivací pro práci na projektu.

Obsah

1 Úvod

1.1 Představení práce

Nápad vytvořit tento projekt vznikal delší dobu, kdy jsem nejprve uvažoval o realizaci jedné z mých oblíbených deskových her, postupně jsem ale začal o problematice realizace uvažovat obecněji a následné zjištění, že v dnešní době neexistuje žádná možnost, jak by si mohl člověk bez znalosti programování vytvořit deskovou (nebo jinou) hru, která by byla současně volně šiřitelná. Mě společně s možností pokusit se takovouto platformu navrhnout a vytvořit jako Bakalářskou práci motivovala natolik že jsem se rozhodl pro jeho realizaci.

Motivací pro vytvoření práce byla také touha vyzkoušet si své vlastní návrhářské schopnosti a obliba deskových her, jež v dnešní době díky rozvoji zábavního průmyslu zaměřujícího se na počítače, herní konzole a podobná zařízení pomalu upadají.

1.2 Cíl práce

Na základě studia existujících deskových her jsem se rozhodl vytvořit návrh a realizaci interpretu deskových her, včetně uložení reprezentace herního materiálu, se kterým bude interpret pracovat. Součástí popisu herního materiálu musí být jak jednotliví hráči a herní díly, tak i samotná organizace herního materiálu.

Výsledná realizace herního materiálu musí být dále použitelná interpretem deskových her. Samotný interpret musí popisovat pravidla hry takovým způsobem, aby umožnil tvorbu deskových her i uživateli který nemá znalosti z programování.

Sekundárním cílem projektu je realizovat projekt tak, aby bylo možné uložit data do podoby čitelné člověkem a přenositelné mezi platformami.

1.3 Stručný obsah

Teoretický úvod. V první kapitole si nejprve projdeme pro projekt důležité teoretické znalosti. Součástí kapitoly je analýza existujících deskových her a shrnutí existujících realizací deskových her a jejich vlastností.

Návrh řešení. Tato kapitola popisuje návrh projektu. Celá kapitola je koncipována jako prezentace projektu a čtenář by měl být schopen pochopit z ní princip funkčnosti projektu.

Realizace, experimenty, vyhodnocení. V této kapitole nejprve stručně přestavím realizovanou implementaci jednotlivých navržených částí. Následně je součástí kapitoly ukázka realizace jednoduché deskové hry. Poslední část kapitoly zhodnocuje dosažené možnosti návrhu a jeho aktuální nedostatky.

Závěr. V této poslední části se podíváme na to co se během práce na tomto projektu podařilo, zhodnotíme dosažené výsledky a podíváme se jakým způsobem plánuji pokračovat a projekt dále rozšiřovat.

2 Teorie

Tato kapitola obsahuje dvě podkapitoly. První pojednává o vlastnostech deskových her a jejich členění. Druhá podkapitola shrnuje vlastnosti známých realizací deskových her.

Původním záměrem bylo, aby součástí teorie byly i kapitoly zabývající se uložením dat a existujícími interprety. Začlenění těchto kapitol bohužel nebylo kvůli omezenému rozsahu práce možné.

2.1 Teoretický úvod k deskovým hrám

Desková hra je hra, která zahrnuje počitadla a díly které jsou pokládány, nebo přemisťovány po předpřipraveném herním plánu, dle zadaných pravidel. Deskové hry mohou být založeny na čistě strategických prvcích, na náhodě, nebo na kombinaci obojího. První deskové hry reprezentovaly bitvu mezi dvěma stranami a většina dnešních deskových her má stále za cíl poražení protihráče. V závislosti na čase, výsledné pozici, nebo získaných bodech.

Definice deskové hry převzata z [1], kde je možné také najít další obecné informace o deskových hrách. V případě bližšího zájmu o novodobé deskové hry doporučuji hledat na stránkách boardgamegeek[2], kde naleznete informace i o v české republice neprodejných titulech, ze kterých jsem vycházel.

V této kapitole se seznámíme s vlastnostmi deskových her, které jsem během teoretické přípravy na návrh projektu vypozoroval, během studia pravidel deskových her a při samotném jejich hraní.

Deskové hry obecně jsou relativně jednoduché, jak po stránce pravidel tak po stránce složitosti herního materiálu. Nejsou ale nijak normalizované a autoři jsou limitování pouze možnostmi materiálu ze kterého je hra realizovaná. Zde je třeba dodat že herní principy se velice často recyklují a znovu používají v jiných hrách, což je ryze ekonomický tah tvůrců, nikomu však nebrání vymyslet originální hru.

Deskové hry můžeme prakticky rozdělit na čtyři typy. Každý z typů sebou přináší jisté vlastnosti, které jsou specifické pro všechny hry daného typu. Na následujících řádcích uvádím jednotlivé typy, včetně společných rysů, abych se mohl dále v textu na typy odkazovat.

- Klasické deskové hry (šachy, Carcassone, Osadníci z Katanu)
  - založeny na volbách uživatele
  - nízká míra interakce (cílem hry je hrát sám za sebe)
  - různá míra náhody (kostky, zamíchané balíčky karet, apod.)
  - nezávislé na reálném čase
  - podmínky vítězství jsou dány pravidly
- Party hry (kostky, Citadela, Bang)
  - založeny na volbách uživatele
  - vysoká míra interakce mezi hráči (cílem je hrát proti protihráčům)
  - obvykle vysoká míra náhody
  - nezávislé na reálném čase
  - podmínky vítězství jsou dány pravidly
- Hry založené na lidských znalostech a úsudku (různé kvízy, popiš-namaluj, apod.)
  - založeny na znalostech uživatele
  - různá míra náhody (odvíjí se od interakce hráčů, například při vyhodnocení otázky kvízu, může stejná odpověď od různých hráčů být vyhodnocena odlišně)
  - můžou být závislé na reálném čase
  - velká část je založena na blafování
  - vítězství závisí obvykle na výsledku subjektivního rozhodování hráčů, během hry

- Hry založené na postřehu a rychlosti (UNO)
  - založeny na fyzickém stavu hráče (postřeh a reakce)
  - vysoká míra náhody
  - závislé na reálném čase
  - podmínky vítězství jsou dány pravidly

S ohledem na uvedené vlastnosti jednotlivých typů her je třeba si položit otázku, které hry je možné vůbec realizovat a u kterých to má smysl. Hra totiž může při převodu na počítačovou verzi ztratit své kouzlo. V další části se tedy budeme věnovat pouze hrám které patří mezi Klasické deskové hry a Party hry. Hry založené na postřehu a rychlosti jsou sice realizovatelné, ale realizací ztrácející své kouzlo, alespoň do chvíle dokud bude za standardní periferie považována pouze myš, klávesnice a případně dotyková obrazovka. Posledním nezmíněným typem jsou hry založené na znalostech a úsudku, tyto hry jsou sice také realizovatelné. Dochází ale ke ztrátě kontaktu mezi hráči, který pro většinu z nich bývá klíčovým. Současně se ztrátou kontaktu mezi hráči, umožňujeme hráčům s ohledem na dostupnost informací na internetu podvádět. Z výše uvedených důvodů jsem se rozhodl poslední dva typy deskových her nezařadit mezi vzorek deskových her vybraných pro návrh projektu.

V další části si důkladněji projdeme některé důležité rysy deskových her z pohledu možností jejich realizace. Dále pak následují pro návrh důležité vlastnosti deskových her a nakonec rozbor prvních dvou typů deskových her z pohledu herního materiálu a práce s ním.

2.1.1 Náhodné jevy ve hře

V deskových hrách se obecně nachází relativně velké množství náhodných jevů. Obecně jsem vypozoroval tři hlavní skupiny do kterých spadají všechny mě známé náhodné jevy ve hrách.

První skupinou jsou náhodné jevy vzniklé interakcí hráčů (př. Kámen/nůžky/papír). Vznikají v částech hry kde obvykle závisí na postřehu a reakcích hráčů. Takovéto části her jsou obecně velmi těžko realizovatelné jiným než originálním způsobem, protože realizací takovýchto částí hry nám vznikne výsledek, který neodpovídá originálu, protože část vlivů zanikne (síla hráčů, aj.), část se změní (převod zobrazení hry na obrazovku, aj.) a nové vzniknou(doba cesty dat po síti, aj.).

Druhou skupinou jsou náhodné jevy vzniklé neznalostí hráčů, příkladem může být zamíchaný balíček karet. Každý hráč určí pravděpodobnost s jakou se otočí specifická karta jinak. Tyto jevy předpokládají že hráč nemá v danou chvíli, přehled o celkovém stavu hry. Tyto jevy vznikají zvlášť pro každého hráče a závisí na jeho všímavosti a paměti. Tyto jevy jsou v praxi velice dobře realizovatelné, protože není problém omezit hráčům přístup k určitým informacím o stavu hry.

Třetí skupinou jsou náhodné jevy vzniklé provedením nějaké nedeterministické akce (např. hod kostkami). To zda je možné jev realizovat závisí na tom, zdali je známé rozložení pravděpodobnosti jednotlivých možných výsledných stavů. Pokud jej neznáme vedla by realizace takovéhoto jevu na jev jiný. U této třetí skupiny je třeba si dále uvědomit že obvykle máme při realizaci možnost použít pouze pseudonáhodné hodnoty. Je tedy na místě ptát se, zdali je možné jev pomocí nich realizovat, bez toho aby byl poznamenán výsledek. Během procházení reálných deskových her jsem na žádnou takovou část hry nenarazil, nicméně její existenci nelze vyloučit. Je třeba, ale klást důraz na kvalitu generátoru náhodných čísel. Důležitým parametrem generátoru pseudonáhodných čísel je závislost generovaných N-tic. Obecně platí že generátor může mít závislosti N-tic, pokud se ve hře vyskytuje potřeba použít K-tici náhodných čísel a K je menší než N.

Do projektu jsem se rozhodl zahrnout všechny náhodné jevy z druhé skupiny a část jevů ze skupiny třetí které mají známé rozložení pravděpodobnosti.

2.1.2 Interakce mezi hráči

Interakci mezi hráči nelze specificky popsat, jelikož mnoho her si pro interakci mezi hráči specifikuje vlastní pravidla a vytváří tím specifický herní zážitek. Obecně ale existují tři skupiny, do kterých se dají interakce mezi hráči rozdělit.

- Interakce fyzická
- Interakce definovaná v rámci pravidel
- Interakce verbální a vizuální(psanou)

Dále rozebírat ale budeme jen jednu, protože interakci fyzickou není v současné době možné realizovat s běžnými periferiemi počítačů. Nebudeme se také zabývat interakcí mezi hráči, která je definována v rámci pravidel (obchodování mezi hráči, apod.), protože její řešení je součástí realizace pravidel.

Zabývat se zde budeme interakcí verbální, případně interakcí psanou. Zde při realizaci narážíme na fakt, že zatímco jsme schopni takovouto interakci velice dobře realizovat, je pro nás prakticky nemožné ji naopak omezit. Existují totiž hry kde když začnou hráči, nebo skupina hráčů mezi sebou komunikovat nad rámec pravidel, hrozí riziko že tím hra pozbude smyslu(Battlestar Galactica, Bang, Citadela).

Projekt si klade za cíl umožnit hráčům komunikaci. Neklade si ale za cíl zamezit komunikaci mezi hráči, protože by se jednalo o nemožný cíl.

2.1.3 Závislost na čase

V této části je třeba ujasnit si že, pod pojmem závislost na čase není myšlena herní doba (čas za který je hra dohrána). Valná většina dnešních deskových her nemá v pravidlech žádné na čase závislé části. Existuje ale malé procento her které naopak využívají reálný čas, obvykle se reálného času používá pro vytvoření stresového prostředí a omezení času který má hráč na daný úkon. Případně jako porovnávací kritérium.

Ačkoliv není problém části hry závislé na čase implementovat. Obvykle takovéto hry patří mezi „Hry založené na znalostech a úsudku“, nebo mezi „Hry založené na postřehu a rychlosti“. Tyto hry jsem ale kvůli výše uvedeným důvodům z cílů realizace vypustil. Proto si projekt neklade za cíl realizovat všechny možné časové závislosti.

V Klasických deskových hrách a v Party hrách se závislosti na čase vyskytují zřídka, pro jejich realizaci ale stačí ve valné většině případů odpočet, nebo stopky.

2.1.4 Vlastnosti deskových her

V této podkapitole si projdeme obecné vlastnosti Klasických deskových her a Party her na kterých bude stavěn návrh projektu.

První společnou vlastností je omezení přístupu k informacím o stavu hry. Jak jsem již uvedl u rozboru náhodných jevů, velká část her úmyslně zatajuje před hráči část informací. Díky jejich neznalosti je hráč nucen předvídat (hráč nevidí do karet soupeře, ale ví kolik jich má).

Společně s touto vlastností je spjata i možnost mít několik vizuálních reprezentací jednoho herního prvku. Vizuálních reprezentace herního prvku se může měnit v závislosti na tom, kolik informací má hráč který se na něj dívá k dispozici.

Dále mohou mít jednotliví hráči pouze omezené možnosti interakce s herními prvky. Ve hře obvykle ke změně možností interakce dochází v závislosti na aktuálním stavu hry. Hráč například může do hry vkládat nové herní prvky pouze v určité fázi (např. fáze Nákupu).

Další společnou vlastností deskových her je absence datových typů. Jakýkoliv parametr ve hře je obvykle podvědomě vnímán jako by měl více datových typů současně, obvyklým příkladem je číslo, které je reprezentováno jako řetězec znaků.

S pohledu herního materiálu je třeba zmínit že jeho množství se během hry nemění. Vezmeme-li si ale vzorek různých herních prvků, zjistíme že některé z nich jsou dále členěny na menší části, které mnohdy mají svůj vlastní význam a chování. Fyzický kus herního materiálu se prakticky může stát pouze nosičem jiných herních prvků. Jako příklad je možné uvést díl herního plánu ze hry Osadníci z Katanu.

Poslední podstatnou společnou vlastností zkoumaných her je jejich deterministický průběh, nepočítám-li části kde do průběhu hry mohou zasahovat hráči. Hráč má ale vždy možnost vybrat si pouze s omezeného počtu možností. Průběh hry je tedy možné realizovat například konečným automatem.

2.1.5 Popis částí Klasických a Party her

V této podkapitole si ukážeme jednotlivé entity které se ve hře vyskytují společně s reprezentací vztahů mezi nimi. Entity vyskytující se ve hře lze prakticky rozdělit následovně.

- *Hráč(i)*
- Herní plán
  - Založený na opakující se mřížce (v plánu lze indexovat)
  - Nezakládající se na mřížce (v plánu nejde indexovat)
- Atomický herní materiál
  - Obyčejné figurky
  - Obyčejné karty (Canasta)
  - Žetony
  - …
- Neatomický herní materiál
  - Figurky s proměnnými parametry
  - Karty obsahující herní mechanizmy, nebo interaktivní prvky
  - Části herního plánu
  - …
- Ostatní herní materiál
  - Herní přehledy
  - Pravidla

Z pohledu rozdělení herního materiálu je možné nabýt dojmu, že herní materiál není nijak složitý. Což je na jednu stranu pravda, protože tvůrci jsou omezeni možnostmi materiálu, ze kterého jsou hry vyrobeny. Na druhou stranu je u většiny moderních her prakticky nemožné její herní díly přesně začlenit. Během posledních několika let totiž pomalu docházelo k v vyčerpání možností, které přinesly deskové hry z přelomu tisíciletí jako byli například Carcassonne, nebo Osadníci z Katanu. Tyto hry zaznamenaly masový úspěch zejména díky jejich originalitě a snáze pochopitelným pravidlům. Během následujících let vznikalo mnoho klonů těchto známých her. V současné době je, ale trh těmito hrami přeplněn a začínají se objevovat nové originální hry, jejichž společnou vlastností bývají obvykle složitá pravidla a nové herní mechanizmy.

Právě tyto nové herní mechanizmy komplikují třídění herní dílů, protože mnohdy se v různých částech hry s daným herním dílem pracuje odlišně, nebo zcela novým způsobem. Pokud při třídění zůstaneme na nejvyšší úrovni výše uvedeného členění, je možné do něj zařadit i herní díly z nově vznikajících her. Protože na rozdíl od členění na nižších úrovních, reprezentuje pouze základní chování herního prvku, nikoli jeho fyzickou stavbu.

Nyní se tedy podíváme na jednotlivé herní prvky z pohledu členění na nejvyšší úrovni. Členěním na nižších úrovních se nebudu zabývat, protože by to z pohledu rozsahu práce nebylo možné.

Vezmeme-li si první skupinu do které spadají všechny herní plány, je důležité především zdali je herní plán založen na opakující se mřížce, nebo zdali je pouhým sloučením ať již souvisejících, nebo nesouvisejících částí. Zatímco v prvním případě je důležitá primárně pozice na plánu položených herních dílů(figurek, apod.), v druhém případě je obvykle důležité pouze to zdali na daném místě jsou herní díly.

V případě první skupiny může být samotný herní plán složen z menších dílů, které je nutné teprve do výsledného plánu sestavit. V těchto případech na díly pohlížím nejen jako na součásti herního plánu, ale také jako na samostatné herní díly.

Herní materiál jsem rozdělil na dvě skupiny, zde je nutné vědět, že organizace herních dílů z pohledu hry nemá na členění vliv. První skupinu jsem nazval Atomický herní materiál a její součástí jsou všechny herní díly, které neobsahují pozice na které je možné položit jiný herní díl. Zatímco skupina neatomických herních dílů obsahuje díly na které je možné pokládat jiné herní díly (část mapy ze hry Carcassonne), případně díly které mají komplexní vnitřní organizaci (figurky s otočným podstavcem).

Obecně lze říci že zatímco herní díly z obou skupin mohou mít libovolný počet parametrů. Díly ze skupiny neatomických herních dílů mají navíc určitou vnitřní strukturu.

Posledním typem herního materiálu který se ve hrách vyskytuje jsou herní díly se kterými v rámci pravidel nepracuje. Do této skupiny patří různé přehledy a pravidla. Obecně lze říci že hru je možné hrát i bez dílů z této skupiny.

Z pohledu pravidel existují dva typy přístupu k herním prvkům. V prvním případě nás zajímá pouhá sama existence určitého stavu hry, nebo splnění určité podmínky. Příkladem může být kontrola zdali má hráč dost surovin pro stavbu, apod. U tohoto přístupu nás zajímá pouze daný herní prvek bez ohledu na jeho okolí (nezajímají nás ostatní nepoužívané suroviny, apod.). V druhém případě pravidla udávají v jakém kontextu vzhledem k okolním prvkům se musí herní prvek nacházet. Takováto pravidla jsou mnohem těžší na vyhodnocení, protože možné kontextové podmínky mohou být prakticky neomezené, protože člověk si s nimi dokáže relativně jednoduše poradit, jejich definice v PC ale může být složitější, protože člověk si pravidla optimalizuje tak aby se mu s nimi dobře pracovalo a postupem času se je učí. Pravidla se tak stávají součástí jeho podvědomí (příkladem mohou být Šachy, kde člověk neuvažuje nad tím jak je možné pohnout figurkami, protože možnosti figurek má v podvědomí, ale zaměřuje se na eliminaci figurek nepřítele).

Nyní se podíváme na samotnou strukturu pravidel, specificky na popis průběhu hry. Běžné pravidla se skládají ze tří částí jimiž jsou fáze inicializace(příprava herního materiálu, určení začínajícího hráče), fáze průběhu hry a fáze vyhodnocení (sečtení bodů,ohlášení vítěze). Fáze inicializace a vyhodnocení jsou lineární posloupnosti úkonů které jsou z pohledu složitosti nezajímavé.

Průběh hry naopak bývá realizován cyklem opakujících se kol hry. Samotný cyklus běhu hry bývá ukončen buď po předem daném počtu kol, nebo případně po splnění předem dané podmínky. Zde je třeba také rozlišit zdali hra končí okamžitě, nebo až na konci kola.

Strukturu průběhu jednoho kola (iterace cyklu běhu hry, ve které měli všichni hráči možnost zasáhnout do průběhy hry) je možné popsat za pomoci cyklů s předem známým počtem opakování, nebo z konečnou podmínkou a rozcestí 1zN. Rozcestí mohou být jak deterministická(definována pravidly), tak nedeterministická(interakce hráče).

Z pohledu struktury popisu průběhu hry nejsou pravidla nijak zvlášť složitá. Složitými částmi pravidel jsou jejich lineární části, specificky pak různé dotazy na kontext hry. Vzhledem k tomu že tyto dotazy jsou unikátní pro danou hru a nejsou nikterak omezeny na rozdíl od fyzické stavby jednotlivých herních dílů. Samotnými dotazy na kontext hry a jejich zpracováním se budu zabývat až v kapitole návrhu, protože se jedná o část pravidel jež nelze na úrovni teorie dost dobře popsat.

2.2 Existující realizace deskových her

Ačkoliv cíl projektu je pokud mi je známo unikátní a nenašel jsem žádné jiné projekty, které by se stejnou tématikou zabývaly a měly současně také stejné cíle z pohledu komfortu práce. Existuje velké množství implementací známých deskových her pro počítač. Tato kapitola obsahuje pouze krátké zhodnocení vlastností, které jsou pro tyto projekty společné. Jako index existujících implementací je možné použít například [3,4]. Součástí kapitoly nebude detailní popis jednotlivých implementací, protože to rozsah práce neumožňuje a cílem projektu není realizace jedné specifické hry, ale platformy pro jejich tvorbu.

Ze samotných implementací jsem moc inspirace čerpat nemohl protože jsem v nich nacházel převážně prvky kterých jsem se sám chtěl vyvarovat. Existující řešení je prakticky možno rozdělit na dvě skupiny a to na skupinu implementací snažících se vytvořit prostředí pro jednoho hráče s AI a na skupinu snažících se zprostředkovávat hru více hráčům.

Typickým představitelem první skupiny je spustitelný soubor. Obvykle dostupný pouze pro jeden operační systém. Samotná hra pak bývá realizována dle mého názoru spíše jako experiment, protože většina realizací sdílí následující nedostatky.

Jako nejzávažnější nedostatek považuji fakt že o většinu projektů se po publikování už nikdo nestará a nejsou dostupné ani jejich zdrojové kódy. Díky tomuto je většina takovýchto projektů nerozšiřitelná a zastaralá. Společně s touto vlastností bych jako zásadní nedostatek považoval že autor nepředpokládal rozšiřitelnost projektu v budoucnu, například s příchodem rozšíření hry. Méně zásadní, ale přesto podstatným nedostatkem je obvykle špatné grafické zpracování (problémy s rozlišením), společně s faktem že projekty jsou obvykle realizované pouze v jedné lokalizaci. Vytváří většina projektů pro potenciální hráče nevlídné prostředí. Tyto zde uvedené důvody jsou dle mého názoru společně se špatnou propagací hlavními důvody, proč tyto projekty nejsou příliš rozšířené a nemají valný úspěch.

Představitele druhé skupiny jsou typicky aplikace realizované pro webový prohlížeč, nebo smartphone. Na jednu stranu nám tento způsob realizace umožňuje mít vždy aktuální verzi, na druhou stranu je ale potřebné mít obvykle trvalé připojení k internetu. Představitelé této skupiny totiž převážně neobsahují AI a jsou stejně jako představitelé předchozí skupiny pevně svázáni s jednou jazykovou lokalizací a jedním vzhledem. Jejich celková kvalita zpracování je ale na vyšší úrovni.

Zvláštní odstavec bych věnoval AI v existujících hrách. Ačkoliv jsem neměl možnost se na realizaci AI podívat ve zdrojových textech. Obvykle je pro běžného uživatele nepřijatelně obtížné AI porazit. Pokud by měly hry možnost nastavit obtížnost AI, nebyl bych toto považoval za problém, bohužel ale většina z nich tuto možnost nemá. Navíc v některých realizacích jsem si všiml že se AI drží stále stejné strategie. Jednalo se ale o hry kde k dosažení vítězství není třeba dlouhodobě plánovat, takže se nejednalo o vážný nedostatek. Hry ve kterých je třeba plánovat dlouhodobě dopředu jsem s realizovaným AI nenašel vůbec.

3 Návrh řešení

Cílem této kapitoly je na základě existujících postupů a znalostí nabytých analýzou existujících deskových her, vytvořit ucelený návrh projektu, protože originální zadání je pro potřeby návrhu projektu příliš abstraktní.

Celý návrh projektu jsem se rozhodl postavit na modelu MVC. Má práce sice realizuje pouze části Model (reprezentovaný uložením dat) a Controler (reprezentovaný interpretem), mým cílem je ale na projektu pokračovat i po odevzdání a projekt dále rozšiřovat a chybějící část doplnit.

3.1 Motivace návrhu

Cílem návrhu je vytvořit interpret deskových her včetně definice uložení herních dat. Předtím než se začneme zabývat samotným návrhem je třeba říci proč projekt vznikl, když v dnešní době existuje mnoho nástrojů kterými lze deskové hry popsat. Základním cílem projektu je umožnit běžnému člověku, jež je neznalý programování, navrhnout a realizovat svou vlastní hru. Existující nástroje jako jsou například programovací jazyky vyžadují pro efektivní využití jejich potenciálu nemalé znalosti a množství zkušeností. Díky čemuž se stávají pro neznalého člověka nepoužitelné, protože ten svou snahu spíš vzdá dříve, než by dospěl k cíli. Proto jsem se rozhodl navrhnout tento projekt tak, aby míra potřebným znalostí nad rámec znalostí hry samotné, byla co možná nejmenší.

Jako příklad bych zde rád uvedl dvě možnosti jak vytvořit textový dokument. První možností je použít LaTeX, ve kterém dosáhnete lepších výsledků, ale k dosažení určité kvality je potřeba mít nemalé znalosti. Na druhé straně existuje Word v němž nedosáhneme na profesionální úroveň LaTeXu, výsledek bude ale uspokojivý. Stejným směrem se snaží jít i můj návrh a to vytvořit uživatelsky nenáročnou alternativu pro realizaci deskových her, k existujícím profesionálním řešením. Cílem projektu rozhodně není znovu vymýšlet existující věci, nebo reimplementovat existující alternativy.

Dále bych rád vysvětlil proč jsem se rozhodl pro deskové hry. Mým dlouhodobým záměrem je studium algoritmů pro automatizovanou tvorbu umělé inteligence (AI), pro problémy kde známe všechny možné volby které může AI provést. Pro úspěch je ale nutné plánovat dopředu a přizpůsobovat svou strategii, dle voleb ostatních entit které se snaží dosáhnout stejného cíle. Své snažení jsem chtěl podložit velkým vzorkem takovýchto problémů a díky mé zálibě jsem je našel v deskových hrách. Rozhodl jsem se nejprve realizovat editor a interpret deskových her, čímž vytvořím smysluplnou aplikaci ve které bude možné algoritmus použít a testovat. Součástí bakalářské práce je jen interpret a úložiště dat. Editor není součástí práce, ačkoli byl původně plánován jako součást projektu, protože stejně jako v případě Wordu se jedná o složitý nástroj, protože realizuje velké množství vizuální a funkcionální abstrakce a v době odevzdání práce byl ve fázi experimentálního návrhu, protože se ukázalo že samotný interpret a uložení dat bylo po stránce návrhu složitější než jsem předpokládal a zabralo tak více času.

3.2 Cíl návrhu

Cílem návrhu je vytvořit definici popisu struktury herních prvků a pravidel, tak aby ji bylo možné na jejich základě hru interpretovat.

Z pohledu reprezentace herního materiálu je cílem navrhnout datovou strukturu, která by umožnila popsat herní materiál za pomoci minimálního počtu typů tříd. Z pohledu práce je třeba, aby bylo možné s datovou strukturou pracovat jako s celkem (dotazovat se na obsažené prvky), současně ale musí být umožněno pracovat s jednotlivými uloženými prvky nezávisle na umístění v datové

struktuře. Uložené herní prvky a jejich organizace musí být schopna reprezentace libovolného stavu hry. Jako úložné médium datové struktury musí sloužit soubor XML, do kterého bude možné jak ukládat, tak z něj načítat.

Z pohledu reprezentace pravidel hry je cílem projektu vytvořit realizaci pravidel tak, že jejím základem je blok obsahující lineární posloupnost instrukcí měnících stav dočasných proměnných, nebo datové struktury hry. Společně s množinou podmínkových skoků, které určují který blok se bude zpracovávat dále. Bloky samotné musí být nezávislé na kontextu běhu hry. Samotný kontext hry, zpracování jednotlivých bloků a interakci s okolím zajišťuje interpret.

3.3 Návrh jednotlivých částí projektu

Nyní když už víme že projekt je vytvářen s ohledem na jednoduchou tvorbu. Seznámíme se z částmi které bylo třeba navrhnout. V základu můžeme rozdělit návrh na část zabývající se návrhem popisu herních prvků a na část zabývající se návrhem reprezentace pravidel. Postupně si tedy projdeme nejprve návrh realizace prvků které se vyskytují v reálných deskových hrách a následně bude následovat část zaměřená na návrh reprezentace pravidel. Obsah celé kapitoly je koncipován jako prezentace projektu, pro člověka co se s ním setkává poprvé.

Z pohledu prvků které se vyskytují v deskových hrách bylo třeba navrhnout reprezentaci hráče a herního materiálu. Společně s nimi jsem navrhl organizaci dat tak, aby reprezentovala skutečné členění herního materiálu v deskových hrách. Při návrhu byl kladen důraz na to, aby v případě že by uživatel musel pracovat přímo s těmito navrženými částmi byla práce z jeho pohledu co nejjednodušší.

V části zabývající se pravidly hry jsem navrhl princip rozdělení pravidel na jednotlivé lineární bloky a definoval jejich propojení tak, aby bylo za pomoci minima takovýchto bloků možné sestavit celé pravidla hry.

3.3.1 Návrh reprezentace herního materiálu

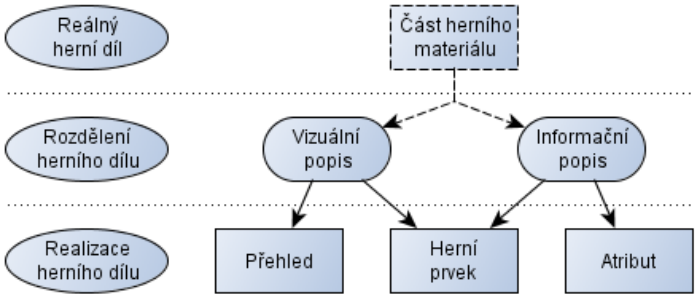
Při návrhu reprezentace herního materiálu byl kladen důraz na obecnou použitelnost třídy která bude herní materiál reprezentovat. Záměrně jsem nechtěl vytvářet specifické třídy pro jednotlivé typy herního materiálu(karty, kostky, figurky, díly mapy, aj.), protože uživatel bude muset prakticky definovat jaké prvky má výsledná realizace hry obsahovat a platí že čím více voleb má tvůrce, tím spíš zvolí špatně. Rozhodl jsem se tedy navrhnout reprezentaci tak, aby se absolutně oprostila od fyzické reprezentace herního materiálu, což umožnilo zjednodušit její vnitřní strukturu. Tímto krokem ale vznikla potřeba definovat navíc odděleně vizuální podobu herního materiálu. Tuto práci návrh přenáší na editor, v čemž osobně vidím přednost, protože je mnohem snadnější definovat jak vizuální tak funkční podobu herního materiálu v Drag&Drop editoru, než na úrovni zdrojových kódů. Editor totiž na rozdíl od práce s kódem může tvůrce tvorbou herních prvků vést krok za krokem a jejich skutečnou reprezentaci vytvořit následně sám.

Samo oddělení vizuální a funkční (datové) reprezentace herních dílů má dvě pro mě zásadní výhody kvůli kterým jsem se pro něj rozhodl.

Prvním důvodem je snížení nároků na aktualizace. Respektive dokud si tvůrce nevymyslí nějakou novou kontextovou vazbu mezi herními díly, může kód zůstat nezměněn. Zatímco kdyby byly obě reprezentace svázány, stačila by i drobná modifikace herního dílu a byla by nutná aktualizace zdrojových kódů. Jako příklad bych uvedl díl herní mapy ze hry Osadníci z Katanu. Pokud bychom například chtěli realizovat cesty skrze střed dílu znamenalo by to v druhém případě vytvořit novou reprezentaci takového prvku. Můj návrh takovouto změnu dokáže realizovat bez nutnosti rozšířit zdrojové kódy.

Druhým méně významným důvodem z pohledu možností návrhu, je možnost vytvořit několik vizuálních zobrazení hry nezávisle na funkčním popisu. Je tedy snadné k existující hře vytvářet například nové lokalizace, HD textury, apod. Navíc jsem tím prakticky umožnil aby každý hráč hrál

například v jiném jazyce bez toho, aby to mělo vliv na hru (vyjma problémů jazykové bariéry, ale to není problém realizace, ale hráčů samých).



Obrázek 1: Návrh realizace herních dílů

Z pohledu herních dílů se nám díky rozdělení reprezentací vytvořily tři skupiny. První skupinu tvoří herní díly které nepotřebují funkcionální reprezentaci, jedná se herní přehledy, tištěná pravidla, aj. Ty jsou z pohledu návrhu nezajímavé protože jejich celá realizace bude součástí editoru. Druhou skupinou jsou herní díly které nemusejí mít samostatnou vizuální reprezentaci, příkladem můžou být žetony herních surovin, apod. Herní prvky z této skupiny jsem nazval Atributy a v návrhu mají vlastní podkapitolu. Poslední skupinou jsou herní díly které potřebují jak funkcionální tak samostatnou vizuální reprezentaci. Této skupině se budu věnovat na následujících řádcích.

Vezmeme-li skupinu herních dílů které potřebují jak funkcionální tak vizuální reprezentaci jedná se o většinu herního materiálu, do této skupiny spadá každý herní díl pro který je důležitý kontext hry ve kterém je použit. Příkladem jsou figurky, díly herního plánu, karty, aj. Tyto herní díly pak můžeme dále dělit do dvou podskupin. První z nich představuje atomické herní díly, jedná se o jednoduché herní díly jako jsou figurky, žetony, karty. Tyto herní díly je možné pouze umístit někam do kontextu hry. Druhou podskupinou jsou neatomické herní prvky, typickým příkladem je díl herního plánu ze hry Osadníci z Katanu. Navíc oproti atomickým herním dílům je možné je kombinovat s jinými herními díly. Například na daný díl mapy lze položit žeton s číslem, figurky měst, cest a zloděje. Přidáním každého takového dílu se mění význam a chování daného dílu mapy.

Obrázek 2: Ukázka herního materiálu 1.část

Pro potřeby realizace neatomických herních prvků umožňuje navržená třída rozšířit popis herního prvku o jeho atomické části. Vezmu-li uvedený díl herního plánu je možné pro něj definovat odděleně šest pozic pro cesty a města a po jedné pro žeton s číslem a zloděje. Oddělení vizuální a funkcionální reprezentace zde ale způsobuje komplikace. V reálném herním prvku jsou tyto reprezentace spojeny takže je zcela jasné kde se na herním prvku jeho atomické části nachází (pozice zloděje ve středu atd.). V mém návrhu je ale vizuální(fyzická) reprezentace oddělena a vzhledem k cíli umožnit její úplnou separaci, ji nelze použít pro potřeby definice umístění jednotlivých

atomických částí, při interpretaci hry. Proto bylo potřeba navrhnout dodatečný funkcionální popis který by umožnil popsat fyzickou stavbu herního prvku.

Jako příklad ukazuje obrázek 2 čtyři karty ze hry „Magic the gathering“[5]. Zatím co první dvě (zleva) karty reprezentují pouze jednu možnou akci a mohou tak být realizovány jako Atomický herní prvek, druhé dvě mají možnost si vybrat mezi dvěma možnostmi a je tedy vhodné realizovat je jako složený herní prvek. Respektive má-li herní prvek více voleb interakce, můžeme se je snažit rozlišovat na úrovni interakce hráče (Pravé/Leve tlačítko myši, CTRL/SHIFT/ALT) a řešit herní prvky jako atomické. Nebo je možné řešit herní díly jako neatomické a rozlišovat pouze mezi tím zdali byl vybrán samotný herní díl nebo jedna z částí (v případě karet na obrázku 2, by schopnost tvořila atomickou část prvku. Hráč by pak vybral buď kartu samou nebo její schopnost). Mé druhé řešení je na první pohled pro uživatele pochopitelnější.

Jednotlivé atomické části jsem na základě potřeby popsat fyzické vztahy rozšířil o možnost definice fyzických vazeb na okolní prvky. Celkem jsem navrhl čtyři typy vazeb, které reprezentují nejběžnější fyzické vazby mezi částmi herních prvků. Pokud by tvůrce potřeboval vytvořit jinou vazbu mezi prvky může ji realizovat na úrovni kódu interpretu, protože to že uložená data nemají informace o fyzickém tvaru herních prvků neznamená, že tyto informace nemá tvůrce hry! Důvodem proč vznikl návrh vazeb je právě složitost jejich popisu na úrovni interpretu, vzhledem k tomu že je projekt cílený na uživatelskou jednoduchost jsem se rozhodl vytvořit vazby na úrovni popisu dat, protože je jednodušší specifikovat vazby v editoru při tvorbě herních dílů, než nad tím uvažovat v interpretu, při realizaci pravidel. Interpret je navíc primárně určen k popisu pravidel a ne k zajištění vazeb mezi prvky.

Návrh v základu počítá se dvěma základními typy vazeb, první z nich je vazba sloučení. Pokud je touto vazbou sloučen libovolný počet částí herních dílů, chovají se z pohledu interpretu jako jeden. Příkladem může být pole pro cesty na dílu mapy ze hry Osadníci z Katanu, jakmile jsou dva díly přiloženy k sobě pole pro cestu se spojí a chovají se jako jeden prvek. Druhým typem vazby je vazba sousednosti pokud jsou touto vazbou spojeny dvě libovolné části herních prvků je možné se následně jedné části dotázat na prvek druhý. Jednoduše je tak možné například zjistit zdali je možné ve hře Osadníci z Katanu postavit na daném místě město. Důležitou návrhovou vlastností vazeb je fakt že jsou si vědomy ostatních vazeb. Například díly spojené vazbou sloučení se chovají jako jeden i pro potřeby vazby sousednosti.

Vazbami se kterými prozatím v návrhu nepočítám jsou vazby pevného spojení a návaznosti. Vazba pevného spojení představuje fakt že dva díly jsou ve skutečné hře spojeny nerozebiratelným spojem. Příkladem můžou být díly mapy které se skládají z bloků sedmi šestiúhelníků, nebo jeden díl domina, jež tvoří dvě části. Vazba návaznosti je alternativou k vazbě sousednosti, na rozdíl od ní ale není určena pro dotazování na sousedy, ale pro algoritmy vyhledávání cesty, apod.

Tyto vazby nejsou součástí realizace protože současný návrh vazeb je pouze testovacím řešením. Na výsledcích práce s ním chci vytvořit finální systém vazeb.

Pro demonstraci uvádím i ukázku herního materiálu ze hry „MIL“[6] (čti 1049). Na obrázku 3 je vidět figurka rytíře, která se bude pohybovat po mapě a bude třeba ji tedy realizovat jako herní prvek, vzhledem k tomu že nemá další funkcionalitu a důležitá je jen její pozice, může být realizována jako atomický herní prvek. Dále zde máme speciální kostku která může být realizována v rámci popisu pravidel. Žetony surovin jež mohou být realizovány jako atributy a dva herní přehledy, jež stačí pokud budou součástí zobrazení hry.

Obrázek 3: Ukázka herního materiálu 2.část

3.3.2 Návrh reprezentace Atributů

Pod pojmem Atribut se v návrhu vyskytuje třída schopná popsat vlastnost herního prvku (barva, cena, jmenovka, apod.) a jejím primárním účelem je rozšíření popisu herního materiálu, právě o možnost přidat jednotlivým reprezentacím herních prvků potřebné parametry. Jak ale bylo uvedeno v návrhu herních prvků, existují případy kdy stačí funkcionální popis herního prvku, jedná se například o herní suroviny. Takovéto herní prvky je sice možné realizovat jako běžný herní prvek. Obvykle ale takovýto herní materiál stačí reprezentovat pouze jeho identifikátorem a množstvím na což stačí i Atribut. Vzhledem k tomu že práce s ním je z pohledu interpretu mnohem jednodušší je vhodné pro reprezentaci takovýchto herních dílů použít právě Atribut.

Samotný Atribut jak již bylo řečeno nemá svou vlastní vizuální reprezentaci, je ale možné jeho hodnotu zobrazit jakou součást herního dílu, který grafickou reprezentaci má (například jako součást herního přehledu).

Když jsem třídu Atributu navrhoval snažil jsem se aby se co nejvíce podobala atributům reálných herních prvků a práce s ním byla z pohledu uživatele co nejjednodušší. S ohledem na tyto předpoklady jsem se rozhodl nevytvářet Atributy s datovým typem, ale místo nich jsem navrhl pouze jeden Atribut, který má z pohledu tvůrce hry skrytý datový typ, který se nastavuje podle typu hodnoty při inicializaci. Tvůrce si pak může hodnotu vyžádat v libovolném datovém typu. Prozatím jsem navrhl jako interní datové typy celočíselné a řetězcové hodnoty.

V případě že je potřeba provést logickou nebo aritmetickou operaci a vstupní hodnoty mají jiné datové typy, provádí se přetypování dle typu prvního a dle účelu operace. Díky tomu může s atributy pracovat i tvůrce který je neznalý datových typů. V případě že se uživatel dopustí operace která je neimplementovatelná (násobení nečíselných řetězců), tvůrci je tato skutečnost oznámena.

3.3.3 Návrh reprezentace organizace dat

Pro návrh reprezentace organizace dat byl nejdůležitějším parametrem jednoduchost z pohledu tvůrce hry. V dnešní době existuje mnoho realizovaných datových struktur, které dostatečně vyhovují potřebám, které existující deskové hry vyžadují. Problém by ale byla jejich prezentace tvůrci hry. Předpokladem návrhu je že tvůrce nezná IT význam pojmů jako je pole, vektor, mapa, apod. Proto bylo třeba skutečnou realizaci organizace dat zapouzdřit do abstraktních kolekcí, které mají pro tvůrce pochopitelný název a chování.

Návrh zvažoval z tohoto pohledu dvě alternativy, první z nich bylo navrhnout abstraktní kolekce herní plán, balíček karet, hromada karet, apod. Tento návrh se ukázal z pohledu použitelnosti stejně nepraktický jako návrh herních prvků se spojenou funkcionální a vizuální reprezentací. Proto jsem se rozhodl navrhnout několik parametrizovaných abstraktních kolekcí, které jsou schopné změnou svého nastavení realizovat několik různých organizačních schémat známých z existujících her. Pro návrh každé z abs. kolekce bylo důležité co za prvky má obsahovat a počet dimenzí jež je potřeba pro popis vnitřní struktury.

Celkem jsem navrhl tři takové abs. kolekce, díky kterým je možné popsat mě známé struktury herního materiálu existujících her. Stejně jako v případě návrhu reprezentace herního materiálu i zde počítám s přenesením starostí o konfiguraci na editor. Předpokladem je že je snadnější krok za krokem v editoru nastavit chování kolekce, než implementovat novou strukturu která by splnila všechny požadované nároky.

První navrženou abs. kolekcí je jedno-dimenzionální kolekce herních dílů, kterou jsem nazval Sada. Účel jejího návrhu je vytvořit strukturu která by byla schopna popsat libovolnou množinu herních prvků, které buď nejsou seřazené a nebo jsou seřazeny tak, že pro každý prvek je důležité pouze kdo je před ním a za ním. Návrh vycházel primárně z různých balíčků karet a neseřazených množin herních prvků. Aby dokázala struktura sloužit více účelům je možné jí nastavit jakým způsobem mají být řazeny data uvnitř a jakým způsobem je do ní možné vkládat a vybírat prvky. Návrh původně zvažoval nechat pouze definici řazení v takovém případě by ale nebylo možné provádět v editoru sémantickou kontrolu. Mimo definice přístupu má struktura definováno také několik operací které jsou pro reálné struktury které popisuje typické, příkladem může být zamíchání prvků (karet).

Druhou navrženou abs. kolekcí je dvou-dimenzionální řídké pole herních dílů, které jsem nazval Mapa. Účel jejího návrhu bylo vytvořit strukturu, která by dokázala celistvě reprezentovat herní plán. Návrh vycházel z herních plánů které jsou složeny z dílů stejného tvaru, prakticky je ale možné realizovat i herní plán, který není organizován podle opakující se mřížky. Stejně jako u předešlého návrhu je možné definovat omezení vkládání, přesunu a odebírání prvků. Co se týče vnitřní organizace a adresování je možné při vytvoření definovat mřížku do které budou jednotlivé prvky vkládány. Návrh definuje mřížku rovnostranných trojúhelníků, rovnostranných šestiúhelníků a čtverců (jakožto jediných pravidelných tvarů schopných vytvořit celistvou plochu). Společně s nastavením mřížky se nastaví i osy dimenzí (a relativní indexy sousedních prvků). Strukturu je možné případně použít bez specifikace definice mřížky, v takovém případě je adresace zcela v rukách tvůrce. Tvůrce se tím ale připravuje o možnost nechat kolekci automaticky vytvářet vazby mezi herními prvky.

Jak jsem již popisoval v 3.3.1 je možné mezi atomickými částmi herních prvků vytvářet vazby jež popisují jejich fyzické vztahy (spojení, sousednost). Zatím bylo možné tyto spojení vytvářet pouze manuálně, to může být velice pracné, vezmeme-li že i relativně malý herní plán Osadníků z Katanu obsahuje takovýchto vazeb 317, počítám-li obousměrné vazby jako jednu. Při tak velkém množství hrozí že by mohlo dojít v velice těžce odhalitelné chybě a mohlo by docházet k nekorektnímu vzniku a zániku vazeb, vlivem chyb tvůrce. Proto jsem do návrhu začlenil možnost definovat pravidla pro automatické vytváření a rušení těchto vazeb. V případě Osadníků z Katanu se jedná o 36 pravidel pro libovolně velký herní plán.

Pro vyhodnocování pravidel je navržena optimalizace kdy pravidla vytvářející vazby v rámci jednoho herního dílu jsou vyhodnocovány pouze při vstupu a výstupu ze struktury.

Při přesunu je zajištěno že se nejprve zruší původní vazby a na novém místě se poté vytvoří nové. Pravidla využívají definované mřížky, aby určily pozice sousedních prvků. Při samotném vyhodnocování pak platí, že pravidlo je přeskočeno pokud soused neexistuje, nebo nemá definovanou atomickou část herního prvku. Toto chování umožňuje mít ve struktuře více různých typů herních prvků bez toho, aby pravidla pro tvorbu vazeb způsobovala problémy.

Třetí navrženou abs. datovou strukturou je jedno-dimenzionální množina jiných abstraktních datových struktur. Navržena byla pro potřeby sjednocení několika jiných struktur do logického celku a jejím cílem není popis organizace dat, ale zpřehlednění výsledného popisu organizace herních prvků.

Z pohledu návrhu sdílí vlastnosti s první uvedenou abs. datovou strukturou jediným rozdílem jsou prvky které se do struktury vkládají.

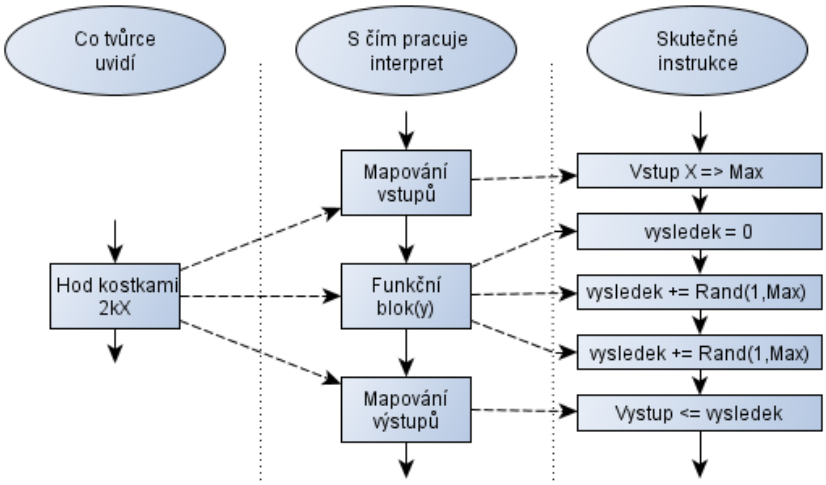
3.3.4 Návrh reprezentace hráče

Z pohledu hry je reálný hráč zajímavý, protože ve své podstatě je třeba navrhnout systém vlastnictví herních prvků které mu náleží a současně definovat jeho interakci se hrou. Z pohledu reprezentace vlastnictví je hráč realizován třídou která může obsahovat libovolný počet mnou navržených abstraktních datových struktur, společně s libovolným počtem atributů.

Třída reprezentující hráče obsahuje pouze data, protože interakce se hrou a ostatní informace o hráči jsou součástí interpretu, včetně informací o aktuálně hrajícím hráči, apod.

3.4 Návrh realizace pravidel

V této části se budeme zabývat návrhem interpretu a popisem pravidel do podoby pochopitelné pro interpret. Při návrhu interpretu jsem se inspiroval objektově orientovaným editorem Omnet++. Jeho základem jsou funkční bloky s definovanými vstupy a výstupy. Tyto bloky lze následně propojovat mezi sebou a případně z množiny takto propojených bloků vytvořit blok nový. Mým cílem je, aby reprezentace pravidel měla podobnou strukturu. Tím dávám možnost tvůrci před-připravit si bloky reprezentující například rozdání karet, hody kostkou, apod. Cíl návrhu demonstruje obrázek 4.



Obrázek 4: Princip popisu pravidel hry

Jak je z obrázku vidět v návrhu existují dvě úrovně abstrakce. První úroveň představují Funkční bloky které jsou abstrakcí určité lineární sekvence instrukcí a představují jednoduchý herní úkon. Druhou úrovní je poté samotné abstraktní uživatelské instrukce, například „Hod kostkami 2kX“ (součet dvou hodů kostek s X stěnami). Návrh je vytvářen tak, aby uživatel mohl pracovat maximální možnou dobu na druhé úrovni abstrakce, ale byl mu přitom umožněn přístup i na nižší úrovně.

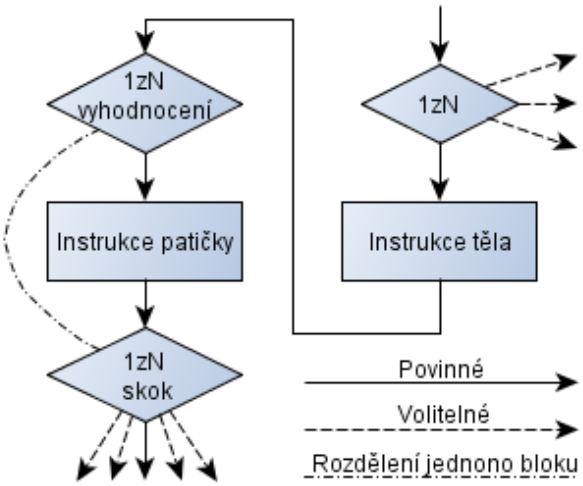
Vzhledem k tomu že uživatelem vytvořené bloky se mohou dále zanořovat do jiných bloků bylo třeba řešit názvy dočasných proměnných. Pro zachování rozšiřitelnosti jsem navrhl interpret tak, aby bylo možné překrývat názvy dočasných proměnných. Principiálně se daná dočasná proměnná hledá nejprve v aktuálně zpracovávaném bloku a následně se prohledávají všechny předchozí zpracované bloky. Dočasná proměnná totiž existuje i po dokončení zpracovávání bloku, až do doby než je zrušena. V případě rušení se postupuje stejným způsobem jako při vyhledávání. Pokud tedy bude autor bloku korektně rušit své dočasné proměnné je možné používat bez problémů bloky od různých autorů.

Samotné funkční bloky pracují s aktuálním stavem hry a dočasnými proměnnými a není možné jim jakkoliv předat parametry a definovat jejich výstup. Proto jak je ukázáno na obrázku 4 je třeba před a za nově vytvořený blok vložit speciální bloky které zajistí přemapování zadaných vstupů na vstupy nového bloku a naopak na konci přemapovat výstupy bloku na očekávané výstupy Respektive řekněme že nový blok očekává vstup v dočasné proměnné A, ale tvůrce má vstupní hodnotu uloženu v dočasné proměnné X. Blok tedy musí vykonat operaci „A ← X“. Z pohledu tvůrce jsou bloky mapování nezajímavé a nestará se o ně, protože je možné je vytvořit v editoru automaticky.

3.4.1 Funkční bloky

Základním důvodem vytvoření návrhu funkčních bloků, bylo vytvoření první úrovně abstrakce, pro usnadnění práce tvůrci. Cílem jejich návrhu je nenutit tvůrce vytvářet velké instrukční bloky. Místo toho je snaha rozdělit je na menší části se snadno pochopitelným cílem, čímž vznikne funkční blok, který představuje pro tvůrce instrukci, která se podobá úkonům prováděných v pravidlech.

Funkční blok je také základním stavebním kamenem popisu pravidel hry a jakožto takový je jediným prvkem se kterým dokáže pracovat interpret. Samotné provádění instrukcí je vykonáno funkčním blokem pokud dostane příkaz od interpretu.



Obrázek 5: Návrh organizace funkčního bloku

Při návrhu funkčního bloku jsem zvažoval dva možné návrhy, prvním z nich umožňoval libovolnou vnitřní organizaci instrukcí a prakticky se podobal procedurám běžných programovacích jazyků. Tento návrh jsem zamítl protože nesplňoval podmínku o uživatelské jednoduchosti. Proto jsem se rozhodl navrhnout blok s pevnou vnitřní kostrou, která bude pro uživatele dostatečně jednoduchá a přesto bude schopna realizovat základní podmínky, skoky a cykly. Návrh vycházel z podmínky že blok musí být schopen realizovat přepínač 1zN, cyklus s podmínkou na začátku a/nebo konci a lineární posloupnost instrukcí v bloku s možnou vstupní podmínkou. Na základě těchto požadavků jsem navrhl následující vnitřní strukturu

Navrženou strukturu funkčního bloku přehledně demonstruje obrázek 5, z kterého lze zjistit že jsem rozdělil funkční blok na čtyři části. Části s instrukcemi obsahují lineární posloupnost instrukcí libovolné délky, případně mohou být prázdné. Pro potřeby reprezentace struktury pravidel a průběhu hry pak slouží dva na sobě nezávislé bloky obsahující podmínkové skoky, pokud je podmínka pravdivá interpret pokračuje zpracováním funkčního bloku na který skok ukazoval (skok může ukazovat i na funkční blok ve kterém je obsažen). Jednotlivé podmínkové skoky jsou v rámci bloku vyhodnocovány dle svých priorit.

Každý funkční blok je vyhodnocován následovně. V případě že je první blok podmínkových skoků prázdný, nebo žádný z nich nemohl být vykonán pokračuje se vykonáním všech instrukcí v těle funkčního bloku. Následně dojde k vyhodnocení druhého bloku podmínkových skoků a jeho výsledek se uloží. Následně vždy nezávisle na výsledku předchozího vyhodnocení dojde k provedení instrukcí v patičce funkčního bloku a poté se provede předtím uložený skok.

Samotný návrh struktury funkčního bloku vychází ze zkušeností s průběhem deskových her a ze snahy vytvořit návrh přijatelný pro tvůrce. Základem návrhu je tělo obsahující instrukce, které realizují funkci bloku. Zatímco možnost vkládat instrukce do patičky bloku byla navržena pro potřeby

úklidových operací před opuštěním bloku, jako je zrušení dočasných proměnných. Co se týče bloků podmínkových skoků, ty jsou navrženy tak aby nebylo nutné zbytečně dělit logické celky pravidel na více funkčních bloků a zachovala se tak přehlednost popisu. Součástí kapitoly implementace je ukázka řešení realizace pravidel jednoduché deskové hry, pro potřeby demonstrace práce s funkčními bloky.

3.4.2 Instrukční sada interpretu a podmínkové skoky

Návrh instrukční sady je prakticky možné rozdělit na dvě části, první je samotný přístup ke zpracování instrukcí, druhým pak je návrh jejich počtu a možností.

V případě první části jsem se rozhodl realizovat instrukce pomocí třídy která nejenom že drží informace o typu a parametrech instrukce, ale je schopná ji i následně zpracovat. Díky tomu že je vše součástí jedné třídy, je možné instrukční sadu rychle rozšířit nebo modifikovat. Samotné instrukce nemají informace o funkčním bloku kterého jsou součástí, mají ale přístup k dočasným proměnným interpretu a k celkovému stavu hry.

Z pohledu návrhu instrukcí byl původní záměr vytvořit instrukce podobné skutečným akcím v deskové hře. Tento návrh ač umožňoval přetěžování vstupů instrukcí byl velmi nepraktický, protože vzhledem k volnosti zápisu a složitosti instrukcí byl výsledný zápis, byť byl krátký, velice nepřehledný. Proto jsem navrhl instrukční sadu kde každá instrukce realizuje jednu operaci se stavem hry, nebo z dočasnými proměnnými. Společně s vyloučením přetěžování a sjednocením formátu zápisu sice výsledný počet instrukcí potřebných pro popis pravidel narostl, stal se ale přehlednějším.

Při návrhu podmínkových skoků jsem vycházel ze zkušeností nabytých při návrhu instrukcí, stejně jako v jejich případě jsem navrhl podmínky tak, aby byly jednoznačné (bez přetěžování) a zapisované v určeném formátu. Samotná podmínka může být složena z libovolného počtu unárních, nebo binárních podmínek, které musí všechny platit, aby se skok provedl. Pokud by bylo potřeba provést logický součet, musí se jeho levá a pravá strana realizovat jako samostatný podmínkový skok.

Jak bylo napsáno výše, návrh obou výše uvedených částí se snaží zachovat jednoduchost instrukcí společně s rychlostí zpracování, na úkor jejich počtu, čili na úkor uživatelské přívětivosti, která je primárním cílem celého projektu. K tomuto jsem se rozhodl na základě dvou předpokladů které mám o výsledné aplikaci. Prvním z nich je předpoklad že se na úroveň instrukcí se bude dostávat tvůrce co nejméně, druhým předpokladem je že pokud k tomu dojde bude editor navržen tak, aby práci co nejvíce usnadnil. Například vzbudil dojem že instrukce mají přetěžované parametry.

3.4.3 Návrh interpretu

Samotný interpret není návrhově nijak zvlášť složitý, neboť většinu práce přenáší na funkční bloky a ty ji dále přenášejí na instrukce. Nejmenší možnou částí reprezentace pravidel kterou dokáže interpret zpracovat je funkční blok, před začátkem hry je třeba nastavit úvodní funkční blok a začínajícího hráče. Dále interpret jen zpracovává jednotlivé funkční bloky. Při návrhu interpretu bylo třeba navrhnout uložení pravidel hry, správu dočasných proměnných, komunikaci interpretu s okolím, držení kontextových informací o hře jež nejsou součástí jejího popisu a správa ostatních prvků potřebných pro běh (generátor náhodných čísel, logovací výstupy, aj.).

Uložení pravidel hry jsem navrhl za pomoci funkčních bloků kde každý z nich má svůj vlastní název kterým je možné blok identifikovat. Současný návrh má všechna jména v jednom jmenném prostoru s čímž nejsem v současné chvíli spokojen a přepracování jmenného prostoru je jedním z mých blízkých cílů. Jméno funkčního bloku slouží k jeho identifikaci pro potřeby podmíněných skoků a inicializace hry.

Z pohledu zpracování instrukcí je interpret navržen tak, aby dokázal pracovat primárně s dočasnými proměnnými, díky tomu je možné si bloky představit jako procedury s konstantními jmény parametrů. Oproti uložení funkčních bloků je s každým navštíveným funkčním blokem vytvořen nový prostor jmen pro dočasné proměnné, takže je možné ve více blocích používat stejně pojmenovanou dočasnou proměnnou. Hledání dočasné proměnné pak probíhá proti směru toku času. Sám interpret spravuje dva různé bloky dočasných proměnných.

Prvním z nich je blok zabývající se správou výše navržených Atributů, který jsem navrhl jako asociativní kolekci referencí. Blok tedy drží buď referenci na některý z existujících Atributů v datovém stromě a vytváří tak k němu přístupový bod, nebo je možné vytvořit instanci která není součástí herních dat a slouží pouze jako dočasná proměnná. Z pohledu tvůrce hry jsou obě možnosti nerozpoznatelné. Tento přístup na jednu stranu přináší komplikace v podobě nutnosti vytvářet alias jména existujícím Atributům, výhodou návrhu je přehlednost díky možnosti pojmenovat si libovolně dočasné proměnné a přehlednost byla při návrhu primárním požadavkem.

Druhým blokem je blok referencí na herní prvky. Blok také používá jmenné prostory stejně jako blok Atributů. Na rozdíl od něj ale neukládají reference na jednotlivé herní prvky, místo nich vytváří zásobníky kolekcí referencí na herní prvky. K této organizaci jsem se rozhodl protože v pravidlech se mnohdy pracuje s množinami herních prvků a použitím kolekcí referencí jsem to umožnil i interpretu, k zapouzdření do zásobníků jsem se rozhodl, proto aby bylo možné reprezentovat zápis pravidel kdy se postupně pracuje s určitými množinami herních prvků, nebo jsou množiny rozdělovány, nebo slučovány pro potřebu následné práce.

Interpret dále drží informace vycházející z průběhu hry a umí s nimi dále pracovat, příkladem je informace o aktuálním hráči. Interpret pak umí aktivního hráče na základě instrukcí měnit.

Poslední částí interpretu jsou části potřebné pro běh hry. V současné chvíli jsou k dispozici logovací výstupy a generátor náhodných čísel. Návrh počítá s tím že je možné libovolnou z těchto částí nahradit jinou se stejným chováním. Příkladem může být výměna generátoru náhodných čísel za kvalitnější.

3.5 Návrh uložení dat

Poslední částí návrhu je návrh uložení realizace deskové hry. Mým cílem bylo navrhnou uložení tak, aby bylo přenositelné mezi platformami a bylo čitelné člověkem. Zvažoval jsem několik alternativ uložení, nakonec jsem se ale rozhodl použít pro uložení realizace značkovací jazyk XML a to kvůli jeho široké podpoře na různých platformách a používanému kódování Unicode, které je schopné ukládat i speciální národní znaky.

Samotný výsledek je prozatím rozdělen na tři části a to na část realizace dat, která obsahuje stav hry, dále na část obsahující popis pravidel a na část obsahující dočasné proměnné ve chvíli uložení. Toto rozdělení jsem navrhl s ohledem na návrhový model MVC, kde první dvě z těchto částí obsahují popis jedné z částí modelu MVC. V případě ukládání hry se pak ukládá jen současný stav hry, dočasných proměnných a kontextové informace interpretu potřebné pro obnovení hry.

Rozhodnutí ukládat data v čitelné podobě sebou ale přineslo několik komplikací, těmi nejvážnějšími jsou problémy s reprezentací odkazů a popisem pravidel. V případě odkazů jsem problém vyřešil, tak že jednotlivé prvky jsou schopny zjistit svou pozici v rámci uložení dat, na místech kde je by pak měl být uložen ukazatel se místo něj do XML ukládá pozice prvku, po načtení je pak možné prvek znovu najít a odkaz opětovně vytvořit. V případě popisu pravidel jsem se rozhodl že všechny instrukce a podmínkové skoky se budou vytvářet z dat v textovém řetězci, nejedná se sice o nejpraktičtější řešení, je ale velmi dobře kontrolovatelné a zakódování a dekódování řetězce není výrazně náročná operace.

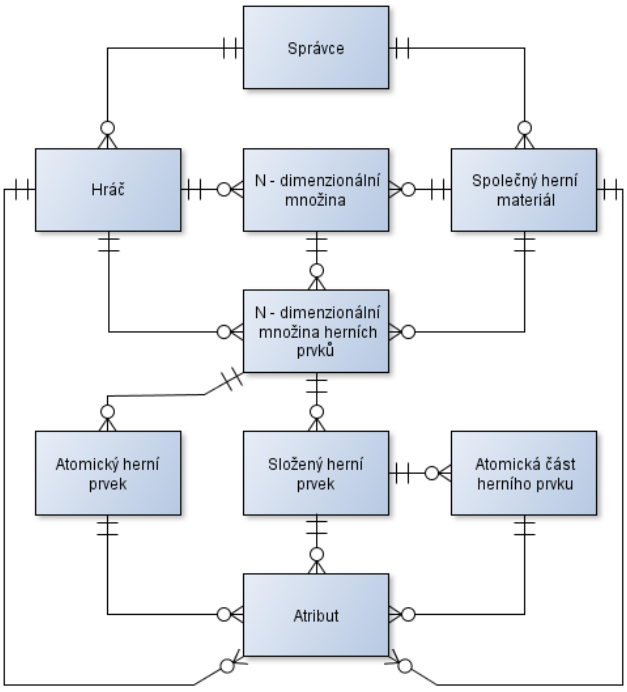
4 Implementace

V této kapitole je stručně popsána implementace projektu, více informací je možné najít ve zdrojových kódech a jejich dokumentaci. Samotná kapitola je rozdělena na dvě části kde první popisuje realizaci uložení reprezentací herních prvků, druhá pak popisuje realizaci interpretu.

Samotnou realizaci jsem se rozhodl vytvořit v jazyce Java, protože umožňuje nejlepší přenositelnost jak spustitelných aplikací tak zdrojových kódů. Při realizaci jsem vycházel převážně z znalostí nabytých během studia a nejvíce pak z předmětu „Seminář Java“. Z knižních zdrojů mi pak nejvíce pomohla kniha Rudolfa Pecinovského[7], kde jsem našel jak správně realizovat návrhové vzory v jazyce Java.

4.1 Realizace uložení herních dílů

Tato podkapitola ukazuje uložení herních prvků, tato část přímo odpovídá části Model návrhového vzoru MVC. Na obrázku 6 můžete vidět navrženou datovou strukturu Jednotlivé entity můžeme rozdělit do následujících skupin. První skupina obsahuje entity „Hráč“ a „Společný herní materiál“ tato skupina představuje entity reprezentující vlastnictví herních prvků. Druhá skupina obsahující obě N-dimenzionální množiny vytváří členění herních prvků. Poslední skupinou jsou entity se kterými pracuje interpret a jsou nosiči dat o stavu hry.



Obrázek 6: Organizace Dat

Herní data jsou uloženy ve stromu kde uzel Správce představuje kořen stromu. Jak je dále možné vidět každý uzel mimo uzlu správce musí mít referenci na otce.

4.1.1 Bázová abstraktní třída

Pro potřeby práce s datovým úložištěm existuje bázová třída, která slouží jako polymorfní rozhraní pro práci s uzly. Její součástí jsou identifikátory uzlu a metody pro vytváření a ukládání uzlu a pro dotazování na potomky. Pro identifikaci uzlu ve stromu se používá vždy jeden ze dvou klíčů. Prvním z nich je uživatelsky definovaný textový a druhým automaticky přidělený číselný.

Uzel dále nese informaci o svém typu. Zjištěna je programem při vytvoření uzlu, nesmí být změněna a uživatel k ní nesmí mít přístup.

Uzel dále definuje metody pro potřeby identifikace uzlu. Pro vyhledávání ve stromu existují metody jež vrací identifikátor „<textový identifikátor><pomlčka><unikátní číselný identifikátor>“ a absolutní cestu k uzlu popsanou těmito identifikátory, identifikátory odděluje tečka, viz. Tabulka.

Voláno entitou	Vrácené jméno	Vrácená absolutní cesta
UzelHráče	uzelHrače-10	spravce-0.entitaHráč-10
UzelMnožiny	množina-60	spravce-0.entitaHrač-15.množina-60
figurka	figurka-80	spravce-0.entitaHrač-20.uzel-50.uzel-70.figurka-80

Tabulka 1: Identifikace uzlů

Pro potřeby uložení entity umí každá vytvořit element XML, který bude obsahovat všechny data entity včetně dat jejích potomků. Každý uzel musí vytvořit minimálně následující element XML.

<dataNode nodeName=*"jméno entity"* nodeType=*"typ entity"* uid="unikátní identifikátor"/>

Každý uzel má možnost vrátit množinu svých potomků, dle zadaného dotazu. Třída definuje nezávisle na sobě dotazy na potomky typu Atribut a dotazy pro ostatní herní prvky. Výsledkem dotazu na Atribut je daný Atribut, nebo null. Výsledkem dotazu na ostatní prvky je vektor výsledků, nebo prázdný vektor. Pro dotazování na potomky musí každý uzel rozumět následujícím dotazům:

Formát dotazu	Výsledek dotazu
* (vyjma dotazů na Atributy)	Vrací se všechny vnořené entity
<jméno entity>	Vrací se entity s daným jménem
<pomlčka><identifikátor entity>	Vrací se entita s daným identifikátorem
<jméno entity><pomlčka><identifikátor entity>	Vrací se entita s daným identifikátorem

Tabulka 2: Dotazy na potomky

4.1.2 Třída Správce

Třída Správce realizuje přístupový bod k datům. Třída je navržena aby plnila funkci přístupového bodu k datům a spravovala uzly vlastnictví (Hráč, Společný herní materiál).

Třída ke spravovaným uzlům přistupuje přes jejich jméno a je zaručeno že názvy uzlů Hráčů nejsou omezeny jmény uzlů Společného herního materiálu.

Pro potřeby funkce přístupového bodu je třída realizována dle návrhového vzoru Singlethon. Dále třída umožňuje nalézt libovolné uzly dle zadané absolutní cesty. Pro tyto potřeby třída obsahuje dvě metody jednu pro dotazy na Atributy a druhou pro dotazy na ostatní uzly. Obě metody vrací vektor obsahující uzly odpovídající dotazu.

Třída rozšiřuje element XML, který ji reprezentuje o své potomky Uzel childs obsahuje potomky uzlu.

```
<dataNode nodeName="!Root" nodeType="ROOT" uid="unikátní identifikátor">
    <childs/>
<dataNode/>
```

4.1.3 Třída Hráč

Pro potřeby reprezentace hráče jsem vytvořil třídu Hráč. Ta sama o sobě může obsahovat libovolný počet uzlů reprezentujících organizaci herních prvků a uzly typu Atribut. Jednotliví potomci jsou identifikováni jménem uzlu a uzel nesmí obsahovat dva potomky se stejným jménem. Uzly typu Atribut mají pro tento účel vlastní jmenný prostor. Pro potřeby vytvoření nového uzlu realizuje třída metody které samy zajišťují vytvoření uzlu a napojení do stromu.

Třída rozšiřuje element XML který jí reprezentuje o své potomky. Uzel attributes bude obsahovat vnořené Atributy a uzel childs ostatní vložené Entity.

```
<dataNode nodeName="jméno hráče" nodeType="PLAYER" uid="unikátní identifikátor">
    <childs/>
    <attributes/>
<dataNode/>
```

4.1.4 Třída Společný herní materiál

Pro potřeby zapouzdření herního materiálu který nepatří žádnému hráči jsem vytvořil třídu Společný herní materiál. Z pohledu uložení dat je třída totožná s třídou Hráč. Rozdíl mezi nimi je až na úrovni interpretu.

XML výstup se oproti entitě Hráč liší pouze hodnotu nodeType která musí být „NONPLAYER“.

4.1.5 Entita Atribut

S ohledem na ulehčení práce tvůrci hry a na základě zkušeností jež jsem nabyl zkoumáním existujících her jsem vytvořil třídu která má možnost reprezentovat číselnou, nebo textovou hodnotu. Implementace umožňuje přidat další datový typ bez vlivu na již realizované projekty.

Třída Atribut umožňuje jako jediná třída popisující uzly vytvořit instanci která nebude napojena do datového stromu herního materiálu a nebude možné ji do něj napojit ani později. Její chování při operacích s instancemi ale musí být stejné jako chování instancí napojených do stromu. Tyto instance jsou používány interpretem jako dočasné proměnné.

Atribut uchovává hodnotu datového typu dle hodnoty kterou byl inicializován. Pro potřeby práce s danou hodnotou jako s jiným datovým typem je prováděno přetypování za běhu interpretu. Atribut je dále schopen vrátit svou hodnotu v každém z datových typů které implementuje.

Třída podporuje logické a aritmetické operace a to v rámci možností i pokud mají vstupní instance jiný interní datový typ. Výsledky aritmetických operací se ukládají do prvního operandu. Aritmetické operace mění pouze hodnotu prvního operandu. Druhý operand zůstává nezměněn. Prvním operandem musí být Atribut, druhým operandem může být Atribut, nebo hodnota v libovolném interním datovém typu.

Pokud je operace nerealizovaná, nebo je nesmyslná, vrací operace hodnotu false. V případě aritmetických operací které nejsou implementovány z důvodu jejich nesmyslnosti, nebo interní chyby

(například při přetypování) zůstávají operandy nezměněny. V následujících tabulkách jsou uvedeny realizované operace.

Logické operace			Výsledek operace	Popis provedené operace
Operand A	**Operace**	**Operand B**		
Řetězec	== , !=	Řetězec	True/False	A <op> B
Řetězec	< , > , <= , >=	Řetězec	False	Nerealizováno
Řetězec	== , !=	Integer	True/False	A <op> (Řetězec)B
Řetězec	< , > , <= , >=	Integer	False	Nerealizováno
Integer	== , != , < , > , <= , >=	Řetězec	True/False	A <op> (Integer)B
Integer	== , != , < , > , <= , >=	Integer	True/False	A <op> B

Tabulka 3: Logické operace

Aritmetické operace			Výsledek operace	Popis provedené operace
Operand A	**Operace**	**Operand B**		
Řetězec	+	Řetězec	True	A = A Konkatenace B
Řetězec	– , * , / , %	Řetězec	False	Nerealizováno
Řetězec	+	Integer	True	A = A Konkatenace (Řetězec)B
Řetězec	– , * , / , %	Integer	False	Nerealizováno
Integer	+ , – , * , / , %	Řetězec	True/False	A = A <op> (Integer)B
Integer	+ , – , * , / , %	Integer	True	A = A <op> B

Tabulka 4: Aritmetické operace

Pokud se interní datové typy neshodují dočasně se vždy přetypuje druhý operand. V případě přetypování si druhý operand zachovává na konci operace svůj interní datový typ a hodnotu.

Pro potřebu exportu do XML rozšiřuje třída XML element dle následujícího vzoru.

```
<dataNode nodeName="jméno entity" nodeType="ATTRIBUTE" uid="unikátní identifikátor"
    dataType="interní datový typ" value="hodnota"
/>
```

4.1.6 Třída Sada

Pro potřeby definice organizace herních prvků jsem implementoval jedno-dimenzionální (1D) množinu. Množina obsahuje herní prvky a pro dotazy na potomky se prioritně používá pozice v množině, je tedy možné mít v jedné množině více instancí se stejným uživatelem definovaným jménem, nikoliv však dvě instance se stejným unikátním identifikátorem.

1D množina má možnost při vzniku specifikovat omezení vkládaní, přístupu a interní organizaci prvků v množině, tyto nastavení nelze změnit po vytvoření instance. Při vkládání do množiny musí metody kontrolovat zadané omezení a případně operaci zamítnout. Při přístupu se nastavení přístupu ignoruje. Je možné se ale na tyto parametry množiny dotázat a odhalit tak neoprávněné operace.

Omezení přístupu	Přístupné prvky (dle pozice, kde N je počet prvků)
FRONT	0
END	N
DOUBLEEND	0 nebo N
RANDOM	0 až N

Tabulka 5: Omezení přístupu k potomkům třídy Sada

Řazení prvků	Popis řazení
FIXATED	Množina udržuje pořadí prvků
RANDOM	Množina neudržuje pořadí prvků

Tabulka 6: Specifikace řazení potomků ve třídě Sada

1D množina musí umožnit realizovat jakoukoliv kombinaci omezení přístupu a řazení prvků (poznámka: při použití řazení RANDOM, je jediným logickým omezením přístupu RANDOM, ostatní nemají smysl).

Pro potřeby vkládání do 1D množiny je třeba realizovat čtyři metody, pro vkládání na začátek, konec a na danou pozici s kontrolou přístupu. Poslední metodou je pak vkládaní na konec bez kontroly přístupu (pro rekonstrukci množiny).

Při přístupu k prvkům pole je třeba rozšířit možné dotazy na potomky tak aby reflektovaly řazení dle pozice. Dotazovací metoda musí rozšířit analýzu první části dotazu před pomlčkou dle následující tabulky, při dotazování se zachovává původní vyšší priorita číselného identifikátoru entity.

Formát dotazu rozšíření možností zápisu první části dotazu	Výsledek dotazu
<jméno entity>	Vrací se entity s daným jménem
<číslo>	Vrací se entitu na daném indexu, nebo prázdný vektor
!First	Vrací se entita na pozici 0
!FirstUnplaced	Vrací se první entita která není položena na jiné entitě
!Last	Vrací se entita na pozici N (kde N je počet prvků množiny)
!Random	Vrací náhodný prvek ležící na některé pozici z rozsahu 0 až N

Tabulka 7: Dotazy na potomky třídy Sada

Pro potřeby vytvoření XML elementu je třeba rozšířit nejen samotný element ale i elementy které obsahuje. Samotné XML totiž nemá specifikované pořadí prvků a jednotlivé prvky nemusejí být uloženy ve stejném pořadí v jakém byly do XML elementu vkládány. Proto je potřeba rozšířit každý obsažený element o informaci o pozici v poli. Typy uzlu jsou „SET“ pro 1D množinu herních prvků a „SETOFSETS“ pro 1D množinu jiných 1D množin.

```
<dataNode nodeName="jméno entity" nodeType="SETOFSETS | SET"
    uid="unikátní identifikátor" inputType="omezení přístupu" outputType="omezení přístupu"
    orderType="řazení prvků" >
    <elements>
        <dataNode … index=<pozice v množině> />
    <elements/>
<dataNode/>
```

4.1.7 Třída 2D mapa

Třída vznikla pro potřeby realizace herního plánu a realizoval jsem jí jako dvou-dimenzionální (2D) řídké pole. Stejně jako v případě třídy Sada je primární přístup přes pozici prvku, proto je možné mít v dané množině více prvků se stejným jménem, nikoliv však číselným identifikátorem.

Oproti Sadě je ale při návrhu kladen větší důraz na okolí prvků. Třída proto má specifikováno jaký tvar mají mít prvky do ní vkládané, prozatím jsem realizoval rovnostranné trojúhelníky, čtverce a rovnostranný šesti-hran(hexagon). To jaký tvar budou mít prvky vkládané do množiny se určuje při jejím vzniku. Dále je možné nastavit maximální absolutní hodnotu indexu dimenzí, při zadání nuly není indexace omezená

Parametr vazby	**Hodnoty**
Typ vazby	SHARED, NEXTTO
Směr vazby	NEW_TO_OLD (vložený prv. –> soused), OLD_TO_NEW (soused –> vložený prv.), DOUBLE_SIDE (oboustrané spoj.)
Jméno množiny Atomických částí herního prvku na vkládaném prvku	Libovolné uživatelem definované jméno
Index do množiny Atomických částí herního prvku na vkládaném prvku	Libovolný uživatelem definovaný index
Index souseda	Index do tabulky relativních vektorů pro definovaný tvar herních prvků, nebo -1 pokud se jedná o vazbu v rámci prvku
Jméno množiny Atomických částí herního prvku na sousedním prvku	Libovolné uživatelem definované jméno
Index do množiny Atomických částí herního prvku na sousedním prvku	Libovolný uživatelem definovaný index

Tabulka 8: Pravidla pro tvorbu vazeb

Třída obsahuje specifikace relativních vektorů ke všem sousedícím prvkům, pro každý realizovaný tvar prvků, ty jsou používány pro operace nad množinou.

Na základě návrhu je potřeba, aby dokázala třída realizovat samočinně tvorbu opakujících se vazeb mezi Atomickými částmi herních prvků. Třída umožňuje definovat nezávisle pravidla pro vznik vazeb v rámci jednoho prvku a pro vazby v rámci prvku a jeho sousedních prvků. Jakmile je vazba definovaná nelze ji zrušit ani měnit její parametry. Třída je realizována tak že se nevyhodnocují vazby v rámci prvku pokud je prvek přesunut na jiné místo v množině. Pro potřeby práce s vazbami

je realizována metoda pro napojení vazeb vně prvku, napojení vazeb s okolními prvky a zrušení vazeb s okolními prvky.

Instance umí pracovat se dvěma typy vazeb a to s vazbou sdílení a vazbou sousednosti. Vazba sdílení definuje že se spojené entity budou navenek chovat jako jeden prvek a to i pro ostatní vazby. Vazba sousednosti definuje že dvě entity spolu sousedí. Každá vazba musí mít specifikovány parametry ukázané v tabulce 8.

Při vyhodnocování pravidel musí platit, že pravidlo je ignorováno pokud Atomická část není součástí jednoho, nebo obou herních prvků, kterých se vazba týká.

Pro práci s prvky má třída metody pro vkládání, odstranění a přesun v rámci třídy. Pokud má množina definované vazby musí se automaticky korektně rozpojit a napojit.

Při přístupu k prvkům pole jsou rozšířené možné dotazy na potomky tak aby umožnily dotaz přes pozici, dle tabulky . Při dotazování se zachovává původní vyšší priorita číselného identifikátoru entity.

Formát dotazu rozšíření možností zápisu první části dotazu	**Výsledek dotazu**
<jméno entity>	Vrací se entity s daným jménem
<X><mezera><Y>*	Vrací se entitu na dané pozici, nebo prázdný vektor

* pro zápis znaménka mínus je použit stejný znak jako pro pomlčku mezi první a druhou částí dotazu.

Tabulka 9: Rozšíření dotazů na potomky třídy 2D Mapa

Při vytvoření XML elementu jsou přidány parametry rozsahu dimenzí a informace o tvaru uložených elementů. Hodnota mapType nabývá hodnot: „TRIANGULAR“, „QUADRANGULAR“, nebo „HEXAGONAL“. V elementu content se uvádějí pouze obsazené pole. XML Element connectionRules obsahuje vazby se sousedními prvky a elementRules obsahuje vazby vně prvku.

```
<dataNode nodeName="jméno entity" nodeType="MAP2D" uid="unikátní identifikátor"
    rangeX="0 až N" rangeY="0 až M" mapType="typ mapy" >
    <connectionRules/>
    <elementRules/>
    <content>
        <position x="index X" y="index Y" >
            <dataNode nodeType="ELEMENT" .../>
        <position/>
    <content/>
<dataNode/>
```

XML formát vazby je následující.

```
<rule conDir="směr spojení" conRul="typ spojení"
    cenSubElemName="jméno množiny Atomických prvků elementu"
    cenSubElemIndex="index Atom. prv. elementu"
    neighborIndex="index do tabulky vektorů sousedů, nebo -1 pro vazbu vně prvku"
    neighSubElemName="jméno množiny Atom. prv. souseda"
    neighSubElemIndex="index Atom. prv. Souseda"
/>
```

4.1.8 Třída Herní prvek

Třída je vytvořena pro reprezentaci herních prvků. Třída rozlišuje zdali se jedná o atomický, nebo složený herní prvek. Složený herní prvek je prakticky rozšířením Atomického herního prvku (figurky, žetony) o možnost popsat složitější herní prvek (například díl herního plánu ze hry Osadníci z Katanu). Typ herního prvku se určí při jeho vzniku a je používán pro optimalizaci práce s instancemi. Prakticky je možné vše realizovat jako Složený herní prvek, zpracování ale bude o něco pomalejší.

Formát dotazu rozšíření možností zápisu první části dotazu	**Výsledek dotazu**
<jméno entity>*	Vrací prázdný vektor
<jméno bloku><mezera><index>	Vrací se Atomická části herního prvku v daném bloku a na daném indexu
<jméno bloku><mezera><hvězdička>	Vrací se všechny Atomické části herních prvků v daném bloku

* jedná se o výjimku, kdy není možnost dotazu přes jméno entity povolena

Tabulka 10: Rozšíření dotazů na potomky třídy Herních prvků

Třída sama obsahuje libovolný počet Atributů přístupných, dle svého jména. Dále třída obsahuje pro potřeby popisu složitých herních dílů, libovolné množství Atomických částí herních prvků. Ty jsou pro přehlednost organizovány po blocích libovolné velikosti identifikovaných pomocí textového řetězce. K jednotlivým Atomickým částem herních bloků (AČ) se v rámci bloku přistupuje indexací. Pro potřeby práce s Atom. část. her. prvky existuje metoda pro vytvoření a inicializaci bloku a metoda pro získání zadaných AČ ze zadaného bloku. Dále jsou implementovány metody pro dotaz na existenci a získání Atom. části herního prvku, ty jsou ale navrženy pro potřeby tříd organizace herního materiálu a používají se pro optimalizaci rychlosti vyhodnocení vazeb mezi AČ, ostatní části projektu by měly používat standardní dotaz na obsažené prvky.

Při přístupu k prvkům pole je třeba rozšířit možné dotazy na potomky tak aby umožnily dotaz na obsažené AČ. Dotazovací metoda musí rozšířit analýzu první části dotazu před pomlčkou dle následující tabulky, při dotazování se zachovává původní vyšší priorita číselného identifikátoru entity.

Pro vytvoření XML elementu jsem přidal parametr definující AČ na kterém je herní prvek položen. Entita musí být identifikována pomocí své absolutní cesty. Parametr placedOn se nevkládá pokud není herní prvek nikde umístěn. Dále je třeba přidat informace o blocích obsahující AČ. K jednotlivým AČ je přidána informace o jejich pozici v bloku.

```
<dataNode nodeName="jméno entity" nodeType="ELEMENT" uid="unikátní identifikátor"
   elementType="COMPLEX | SIMPLE" placedOn="absolutní cesta k prvku" >
   <attributes/>
   <subElementSets>
      <subElementSet setName="jméno bloku" >
         <dataNode nodeType="SUBELEMENT" ... subElementIndex="pozice v bloku" />
      <subElementSet/>
   <subElementSets/>
<dataNode/>
```

4.1.9 Třída Atomická část herního prvku

Třída je navržena pro potřeby realizace menších částí herních dílů. Může mít libovolný počet Atributů, nesmí mít ale dva Atributy se stejným jménem.

Dále musí třída držet informaci o herních prvcích které jsou na reprezentované části herního prvku položeny. Pro potřeby manipulace s položenými prvky existují metody na vrácení všech položených prvků, přidání a odebrání prvku.

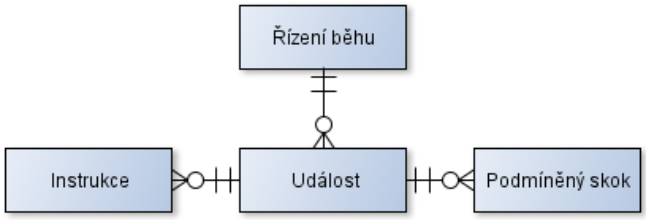
Třída uchovává informace o existujících fyzických vazbách. Třída proto implementuje metody pro přidání a odebrání vazby. Vazba musí být specifikována svým typem a navázaným uzlem. S vazbami se dále pracuje na úrovni třídy Herní prvek.

Pro vytvoření XML elementu je výsledný element rozšířen o informace o vazbách. Pro identifikaci druhého uzlu vazby a položených prvků se používá jejich absolutní cesta.

```
<dataNode nodeName="jméno entity" nodeType="SUBELEMENT" uid="unikátní identif.">
    <attributes/>
    <sharedWith>
        <shared fullName="absolutní cesta k uzlu" />
    <sharedWith/>
    <nextTo>
        <next fullName="absolutní cesta k uzlu" />
    <nextTo/>
    <placedElements>
        <placedElement fullName="absolutní cesta k uzlu" />
    <placedElements/>
<dataNode/>
```

4.2 Interpret

Tato podkapitola definuje interní organizaci a možnosti interpretu. Interpret je založen na postupném zpracování speciálně navržených bloků, které se pro potřeby interpretace chovají jako atomické prvky. Interpret také ukládá záznamy o své činnosti, tak aby bylo možné běh hry později analyzovat.



Obrázek 7: Organizace Interpretu

Obrázek 7 ukazuje základní interní organizaci interpretu. Řízení běhu představuje základní přístupový bod interpretu a tvoří rozhraní běhu interpretu. Událost představuje základní atomickou část popisu běhu hry. Během provádění události nelze běh interpretu zastavit. Událost se pak dále skládá z Instrukcí a Podmíněných skoků.

4.2.1 Řízení běhu

Entita Řízení běhu je navržena jako jádro interpretu. Jejím cílem je nastavení interpretu, držení informací o dočasných proměnných a samotná interpretace Událostí.

Pro potřeby přístupového bodu musí být zajištěno že instance Řízení běhu bude v programu pouze jedna a bude možné k ní přistoupit z kteréhokoliv místa v programu.

Řízení běhu musí dále obsahovat informace o Události která se má zpracovávat a o aktivním hráči. Oba parametry musí být nastaveny před spuštěním interpretu.

Řízení běhu dále obsahuje správu dočasných proměnných, které jsou realizovány jako pojmenované reference na herní prvky, nebo jejich atributy. Řízení běhu hlídá vznik a zánik jmenných prostorů a případně na požádání instrukce vyhledá danou dočasnou proměnnou, nebo jí vytvoří/zruší.

Pravidla jsou uloženy v následujícím XML formátu, kde element rules obsahuje všechny v pravidlech definované funkční bloky.

```
<rules>
    <functionBlock/>
</rules>
```

4.2.2 Funkční bloky

Samotný funkční blok je v realizaci rozdělen na tři části. První z nich je třída reprezentující kostru funkčního bloku, kterou jsem nazval Událost. Ta vytváří komunikační rozhraní mezi Řízením běhu a funkčním blokem, na požádání Řízení běhu zpracovává jednotlivé části funkčního bloku, který reprezentuje. Sama Událost neobsahuje popis posloupnosti jakým se mají jednotlivé bloky zpracovávat. Tento popis je součástí Řízení běhu, který rozhoduje o pořadí v jakém se bloky mají zpracovávat a směřuje běh hry dle výstupů bloků s podmínkovými skoky.

Součástí třídy Událost jsou dva bloky Instrukcí a dva bloky Podmíněných skoků. Jak instrukce tak podmínkové skoky jsou realizovány samostatnými třídami.

Třída realizující instrukce obsahuje typ a parametry instrukce. Instrukci je možno zapsat jako textový řetězec, který je následně dekódován a jednotlivé jeho části jsou uloženy do odpovídající parametrů třídy. Třída samotná obsahuje také potřebné metody pro zpracování instrukcí, pro tyto potřeby má třída přístup k datové struktuře a na může požádat Řízení běhu o libovolnou dočasnou proměnnou. Výsledkem zpracování je hodnota false pokud nebylo možné instrukci zpracovat, nebo true pokud byla instrukce zpracována. V případě chyby při zpracování se obnoví stav hry před započetím zpracování.

Třída realizující podmínkové skoky stejně jako třída instrukcí dokáže načíst zadaný podmínkový skok z textového řetězce a následně podmínkový skok na požádání vyhodnotit. Pro své potřeby má stejně jako třída realizující instrukce přístup k datové struktuře a dočasným proměnným Řízení běhu. Návratovou hodnotou zpracování je výsledek podmínkové části, na základě jejího výsledku je pak možné požádat o cíl skoku.

Funkční blok má následující formát XML. Atribut blockIdentifier udává jméno pod kterým k danému funkčnímu bloku přistupuje interpret. Zatímco atribut blockName udává skutečné jméno funkčního bloku. Instrukce a podmínkové skoky mají stejný formát který je uveden níže.

XML formát bloku:

```
<functionBlock blockIdentifier="identifikator bloku" blockName="jmeno bloku">
    <bodyInstruction/>
    <aftherInstruction/>
    <beforeJumps/>
    <aftherJumps/>
<functionBlock/>
```

XML formát instrukcí:

```
<instruction instructionType="identifikator instrukce" index="pořadí instrukce při zpracování">
    <parameters numberOfParams="počet parametrů instrukce">
        <parameter index="pořadí parametru" value="hodnota parametru" />
    </parameters>
</instruction>
```

XML formát podmínkového skoku:

```
<jump jumpDestination="identifikator cílového bloku" index="priorita vyhodnocení skoku">
    <conditions>
        <condition index="pozice při zpracování" condType="identifikator podmínky"
            paramA="první parametr" paramB="druhý parametr"
        />
    <conditions/>
<jump/>
```

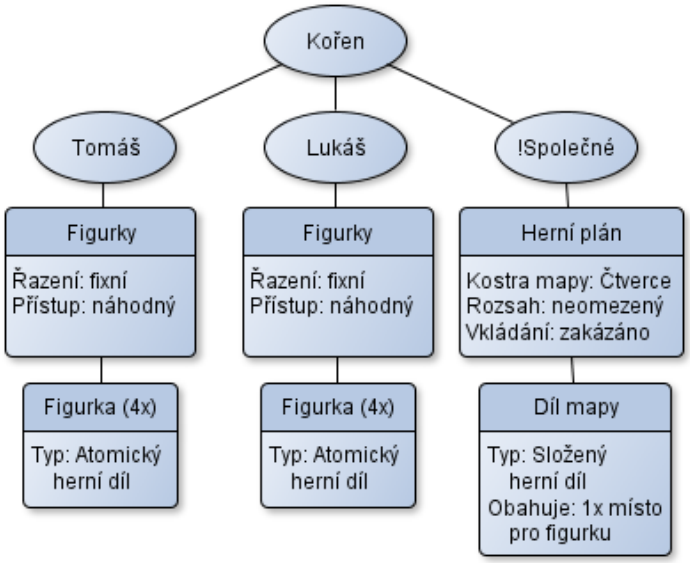
4.3 Ukázka realizace deskové hry

V této podkapitole bude ukázána realizace jednoduché deskové hry „Člověče nezlob se“. Tento příklad má prezentovat postup dosažení výsledku, nikoliv možnosti samotného návrhu. Protože na rozsáhlejší ukázku není místo, navíc realizace v kódu, nebo XML není při větším rozsahu přehledná. Realizace této deskové hry je součástí odevzdaných příkladů.

Seznam všech demonstrativních ukázek, včetně krátkého popisu je součástí dodatků.

4.3.1 Člověče Nezlob Se

Z pohledu dat třeba pro každého hráče vytvořit jeho čtyři figurky, které mohou být realizovány jako atomický herní prvek. Samotná Sada obsahující figurky hráče má fixní řazení prvků, protože je potřeba k nim přistupovat přes index, není třeba ale omezovat přístup k nim. Samotný herní plán je definován s čtvercovým podkladem, bez omezení rozsahu. To si zde můžeme dovolit, protože je do herního plánu zakázáno vkládat nové pole (figurky na existující pokládat lze). Přehledně organizaci dat ukazuje obrázek 8.



Obrázek 8: Člověče nezlob se - organizace herních dílů

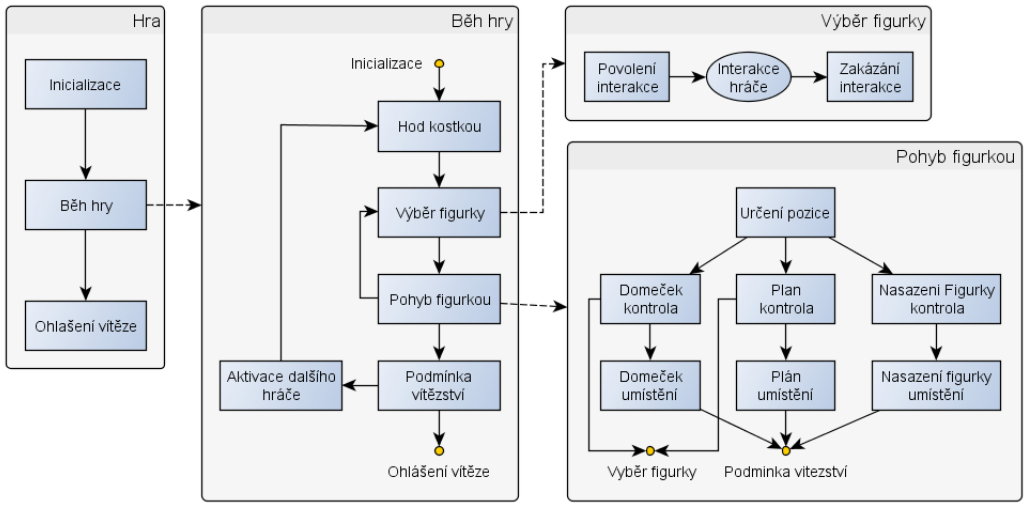
Na obrázku 9 je ukázán příklad návrhu pravidel včetně zapouzdření bloků, které bude prováděno na úrovni editoru. Blok Hra se skládá ze tří bloků, dále se budeme zabývat pouze jedním, protože Inicializační blok je nezajímavý a blok Ohlášení vítěze bude závislý na zobrazení hry. Rozbalením bloku Běh hry se dostaneme na hlavní cyklus hry. Bloky Výběr figurky a Pohyb figurkou jsou složeny stejně jako blok Běh hry z dalších bloků. Ostatní bloky reprezentují už samotné funkční bloky.

Nyní si projdeme realizaci několika funkčních bloků zapsaných v pseudokódu. Jednotlivé funkční bloky jsou rozděleny na samostatné části odpovídající jednotlivým částem funkčních bloků.

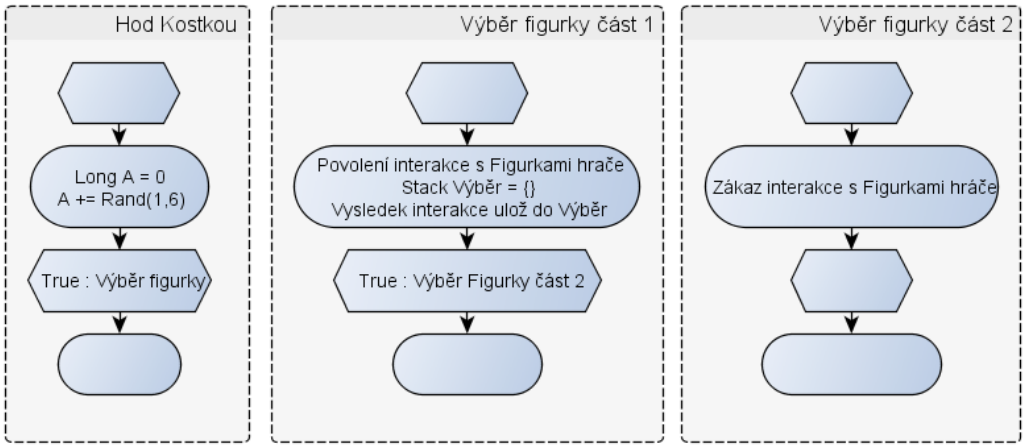
Prvním funkčním blokem je Hod Kostkou (obr. 10). Zde se vytvoří proměnná A reprezentující hodnotu která padla na kostce.

Samotný výběr figurky (obr. 10) je rozdělen na dvě části, které se nezpracovávají ihned po sobě, ale po dokončení první se interpretace zastaví do doby než hráč vybere figurku. V prvním

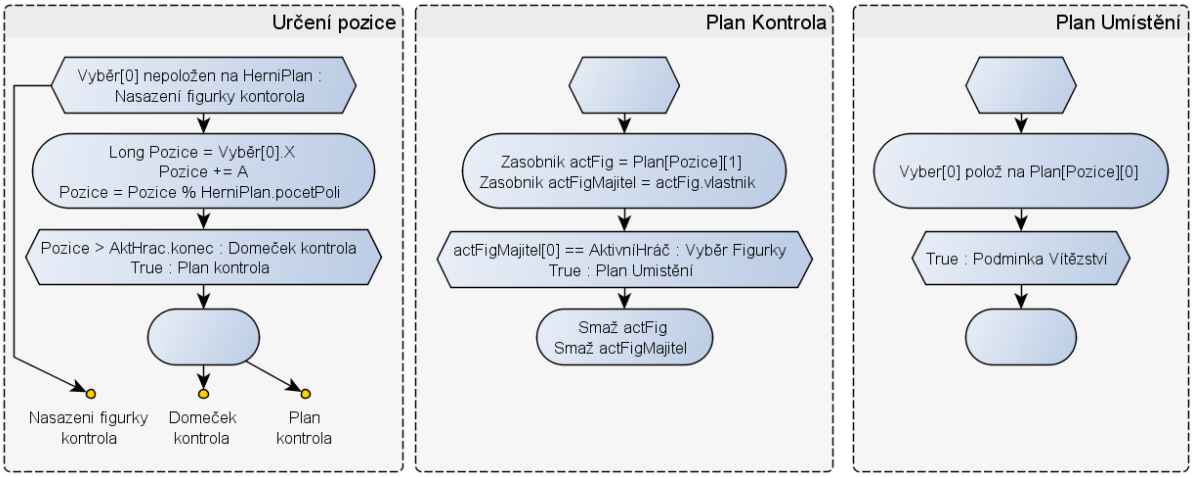
funkčním bloku informuje interpret program o tom se kterými částmi může hráč manipulovat a specifikuje kam se má uložit hráčova volba. Druhý funkční blok jen deaktivuje možnosti interakce.



Obrázek 9: Ukázka Realizace Pravidel



Obrázek 10: Funkční bloky 1.část



Obrázek 11: Funkční bloky 2.část

Jakmile jsme zjistili od hráče kterou figurkou chce hrát, můžeme z toho zjisti co chtěl hráč provést. Ve většině jiných her je ale třeba následně zjisti kam se chce s figurkou pohnout. Ve funkčním bloku výběr pozice lze vidět že pokud hráč vybral figurku, která neleží na herním plánu přechází se k Nasazení figurky, bez toho aby se zbytek bloku zpracoval. Pokud ale figurka leží na herním plánu vypočte se její nová teoretická pozice. Pokud by takovýto pohyb skončil v domečku hráče bude se pokračovat funkčním blokem kontrolujícím zda je na daném místě v domečku volno. Pokud pohyb skončí na plánu pokračuje se funkčním blokem vyhodnocujícím pozici na plánu.

Pokud figurka skončí svůj pohyb na herní plánu zjistí se nejprve zda na cílovém poli nestojí jiná figurka stejného hráče, pokud ano musí hráč vybrat jinou figurku. Pokud je zde figurka jiného hráče, nebo je pole prázdné pokračuje se blokem jež provede přesun.

V bloku umístění figurky na nový díl mapy nejprve dojde k odstranění případné stojící figurky z cílového pole a následně se přesune hráčova figurka. O původně stojící figurku se nemusíme starat, protože je stále součástí Sady figurek protihráče a ten jí může znovu nasadit. Prakticky dojde pouze ke změně hodnoty parametru udávajícího pozici figurky na null.

Další bloky zde už popisovat nebudu, protože se jedná pouze o obměnu již ukázaných bloků. Věřím že tato krátká ukázka byla dostatečná pro pochopení organizace funkčních bloků a realizace pravidel.

5 Dodatky

Na následujících stránce najdete informace o přiložených příkladech, včetně krátkého popisu. Ukázkové příklady jsou na přiloženém DVD ve složce „priklady“, Kazdy příklad obsahuje soubor „navod.txt“, obsahující popis příkladu.

- Příklad demoBP
  - Kompletní realizace herních prvků a jejich organizace pro hru Člověče nezlob Se.
  - Ukázku tvorby pravidel
  - Ukázku funkčnosti ukládání do XML
  - Ukázku funkčnosti načítání z XML

- Příklad uložení organizace herních prvků hry Osadníci z Katanu v XML
  - Výsledný XML soubor
  - Zdrojový kód kterým byl výsledný XML soubor vytvořen

6 Závěr

Během práce na projektu jsem si vyzkoušel návrh a realizaci projektu bez specifického zadání, což osobně považuji za největší osobní přínos. Bohužel se ale také ukázalo že jsem neměl dostatek zkušeností pro tvorbu takovéhoto projektu. Proto bylo třeba různé části projektu několikrát přepracovat, protože se postupně ukazovalo že ač se dané řešení zdálo na první pohled jako efektivní a dostačující. Experimenty s ním ukázaly že bylo navrženo nevhodně.

Prakticky jsem v takové chvíli měl dvě možnosti, nechat řešení tak jak je, pokračovat dál a nedostatky zamlčet. Druhou možností bylo přehodnotit realizaci a případně návrh dané části a vytvořit novou realizaci, která odstraní nedostatky předchozí. Osobně jsem se rozhodl pro druhou variantu protože chci projekt dále využívat ve své další práci.

6.1 Zhodnocení odvedené práce

Z pohledu práce se mi podařilo vytvořit, dle mého názoru, kvalitně navrženou strukturu interpretu a uložení herních prvků. Při jejich realizaci jsem si byl vědom že ačkoliv vycházím z relativně velkého množství her, vždy se najde něco co se současnou verzí vyřešit nepůjde. Proto byla primárním cílem realizace budoucí rozšiřitelnost možností projektu, což se mě dle mého názoru podařilo.

Mám-li zhodnotit aktuální možnosti projektu. Tak považuji uložení herních prvků za strukturálně hotové. V současné chvíli osobně nejsem spokojen s reprezentací vazeb mezi prvky, jinak, ale považuji uložení prvků za dobře navržené a myslím si že jej nebude nutné dále rozšiřovat.

Z pohledu možností interpretu je možné realizovat každou mě známou hru, maximálně s rozšířením instrukční sady, nebo podmínek skoků. Protože nesnažil jsem se během práce na projektu vytvořit kompletní sadu podmínek a instrukcí, ale mým cílem je rozšiřovat je postupně dle potřeby vytvářeného projektu. Z tohoto pohledu uvažuji že kód instrukcí a podmínek uložím mimo kód interpretu, například do jazyka lua, aby nebylo nutné editovat zdrojové kódy při přidávání instrukcí, nebo podmínek.

6.2 Plány do budoucna

Z plánů do budoucna chci v blízké době vytvořit editor deskových her, aby bylo možné začít realizovat složitější deskové hry. Samotný editor je totiž cílem mé práce, protože ač je kód interpretu dle mého názoru dobře pochopitelný, ruční tvorba definice větší hry v XML je prakticky nereálná a právě proto je třeba realizovat editor.

Co se rozšíření současné implementace datové struktury. Plánuji rozšířit prvek Atribut u datový typ enumerace. Dále pak přepracovat systém vazeb mezi prvky, respektive rozšířit funkcionalitu vazeb a umožnit vytvářet vazby i mezi libovolnými prvky.

Interpret pak rozšířit o jmenné prostory identifikátorů funkčních bloků, logování provedených akcí pro možné přehrání záznamu hry. Dále bych rád přidal do interpretu informační značky pro efektivní zobrazení bloků v editoru a také bych rád přidal body obnovy, pro potřeby částí hry kde si hráč najednou rozmyslí své volby a chce začít znovu. Takovéto chvíle je možné nyní realizovat funkčním blokem i inverzními operacemi. Osobně ale z pohledu tvůrce jsou body obnovy jednodušší na práci.

Dlouhodobým cílem projektu je poté co bude hotový editor, začít pracovat na algoritmu pro samočinnou tvorbu AI na základě definovaných pravidel hry. Který považuji za poslední část mého snažení před publikováním projektu pro veřejnost.

Literatura

[1] Kolektiv autorů: Wikipedia Board Game [online]. 11.4.2013 14:58 [cit. 18.4.2013]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Board_game>

[2] Kolektiv autorů: Board Game Geek [online]. [cit. 20.4.2013]. Dostupný z WWW: <http://boardgamegeek.com>

[3] Kolektiv autorů: Online Spielen [online]. [cit. 20.4.2013]. Dostupný z WWW: <http://www.michas-spielmitmir.de/online_spiele.php>

[4] Kolektiv autorů: BrettspielWelt [online]. [cit. 20.4.2013]. Dostupný z WWW: <http://www.brettspielwelt.de>

[5] Garfield, R.: *Magic the Gatering* [karetní hra]. představeno v roce 1993

[6] Firmino M.: *MIL (1049)* [desková hra]. představeno v roce 2011

[7] Pecinovský, R.: *Návrhové vzory* [kniha]. Praha, Computer Press, 2007. ISBN 978-80-251-1582-4. Dostupný v: Knihovna FIT VUTBR

Seznam příloh

Příloha 1. DVD obsahující zdrojové texty, dokumentaci a příklady. Organizace dat na CD je v souboru „OBSAH.txt“