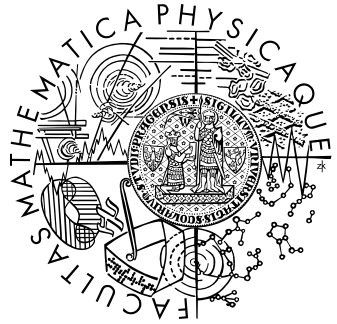
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Všetička

Sokoban

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jiří Dokulil

Studijní program: Informatika, Obecná informatika

2010

Rád bych poděkoval svému vedoucímu práce RNDr. Jiřímu Dokulilovi za pomoc a cenné rady při vytváření a psaní této bakalářské práce. Dále bych rád poděkoval tvůrcům řešitelů kol Brianu Damgaardovi a Ge yongovi za obětavou snahu o pomoc při integrování jejich řešitelů YASS a BoxSearch do mého programu.

Dále bych rád poděkoval své rodině, přítelkyni a přátelům za velkou podporu při studiu.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 31. července 2010 Martin Všetička

Obsah

Název práce: Sokoban
Autor: Martin Všetička
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Jiří Dokulil
e-mail vedoucího: Jiri.Dokulil@mff.cuni.cz
Abstrakt: Cílem bakalářské práce je implementace varianty hry Sokoban a komunitní webové aplikace pro hráče, která jim umožní porovnat své dosažené výsledky. Práce bude zahrnovat:

- dva hrací módy: hru pro jednoho hráče a síťovou hru pro dva hráče,
- řešení kol s využitím existujících solverů a řešení kol od hráčů,
- vytváření vlastních kol a
- možnost naprogramování jednotlivých objektů formou pluginů.

Hra bude napsána v jazyce C#, webová aplikace v PHP nad databází MySQL.
Klíčová slova: Sokoban, síťová hra, pluginy

Title: Sokoban
Author: Martin Všetička
Department: Department of Software Engineering
Supervisor: RNDr. Jiří Dokulil
Supervisor's e-mail address: Jiri.Dokulil@mff.cuni.cz
Abstract: The goal of the Bachelor thesis is to implement a variant of Sokoban game and to create a community website for players which would provide means for comparison of players' results. The work will include:

- two game modes: single player and network game for two players,
- solving of rounds with the help of existing Sokoban solvers and solutions of players,
- creating of rounds and
- possibility of programming particular objects via plugins.

The game will be programmed in C#, the website in PHP and MySQL.
Keywords: Sokoban, network game, plugins

Kapitola 1

1 Úvod

Sokoban (倉庫番 Sōkoban, skladník) je logická hra, ve které hráč ovládá hlavní postavičku, Sokobana, a jejím prostřednictvím posouvá bedny v bludišti a snaží se je umístit na vyznačené pozice. Bludiště je 2D mapa znázorňující rozmístění jednotlivých objektů v bludišti, tj. postavičku Sokobana, bedny, zdi a vyznačená cílová políčka.

Sokoban byl vymyšlen v roce 1980 panem Hiroyukim Imabayshim v rámci soutěže Takarazuckých skladů o navržení motivační hry pro zaměstnance. Zveřejněna byla o dva roky později v roce 1982 softwarovou firmou Thinking Rabbit, jejímž ředitelem byl právě Imabayashi a která sídlí ve městě Takarazuka v Japonsku.

Cílem mé práce bylo vytvořit a integrovat desktopovou aplikaci a webovou aplikaci pro hraní hry Sokoban s vlastnostmi popsanými dále. Pro zatraktivnění aplikace měla desktopová aplikace umožňovat nejen hru pro jednoho hráče, ale i síťovou hru, kdy proti sobě hrají dva hráči a oba hráči vidí jednak svou hrací desku a zároveň vidí i počínání soupeře.

Další součástí mé práce bylo umožnění modifikace hry prostřednictvím samostatně programovatelných herních objektů, čímž se může změnit buďto pouze grafická stránka objektu nebo i jeho chování, a tím již dostáváme jinou variantu hry Sokoban (některé známější varianty hry Sokoban jsou popsány v odstavci 1.2).

Dále měly být do mé práce zabudovány existující řešitelé kol, které mohou pomoci hráčům v nouzi. Jedná se však pouze o řešitele pro standardní variantu Sokobana, jiné varianty nejsou tolik známé a řešitelé se pro ně typicky netvoří. Tyto řešitelé nemusí uspět, a proto je do cíle práce zahrnuta ještě možnost podívat se na řešení jiného hráče.

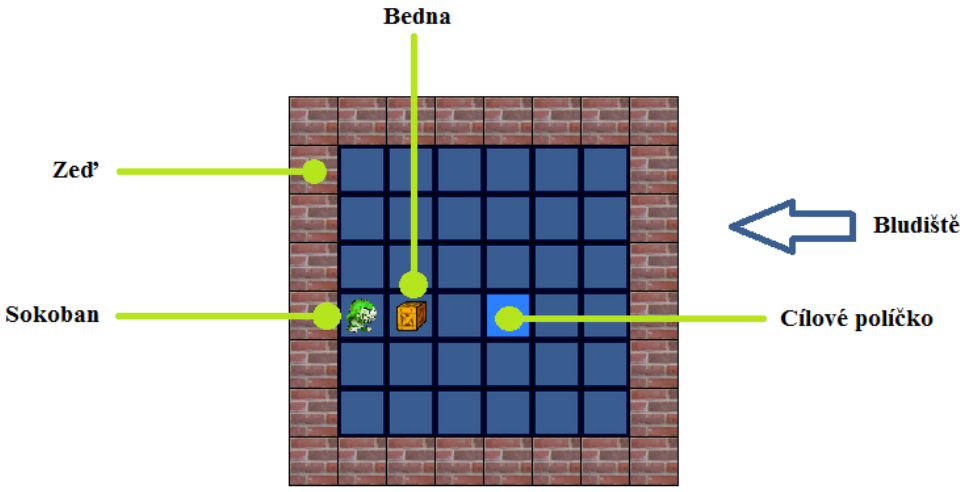
Protože hráči také často vytváří nová kola, měl být součástí práce i editor pro jejich vytváření. Toto je v souladu se snahou o vytvoření integrovaného prostředí, kde hráči nebudou chybět základní náležitosti ke hře.

Webová aplikace měla za cíl zajistit možnost srovnávání výsledků hráčů a poskytnout podporu síťové hře v desktopové aplikaci, to vše za pomocí PHP a MySQL.

Jazyk C#, ve kterém měla být desktopová aplikace naprogramována, měl představovat částečně výzvu v tom smyslu, že jde o manažovaný jazyk, který není pro vytváření her příliš vhodný, protože není možné hlídat paměťové nároky a nelze tolik optimalizovat kód jako v jiných programovacích jazycích produkujících nativní strojový kód.

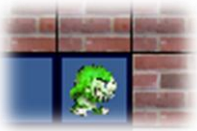
1.1 Pravidla pro standardní verzi Sokobana

Jak již bylo řečeno výše, hra se sestává z bludiště, Sokobana ovládaného hráčem, bednami, zdmi a cílovými políčky. Velikost bludiště není pravidly omezena. Typicky se se vzrůstající velikostí bludiště zvyšuje obtížnost. Příklad velmi jednoduchého bludiště je zobrazen na obr. 1.



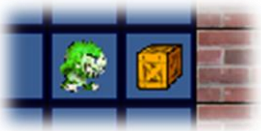
Obr. 1

Sokoban se může pohybovat čtyřmi směry, a to nahoru, dolů, doleva a doprava. Pokud se ve směru Sokobanova pohybu vyskytuje zeď, pak Sokoban zůstane na své pozici. Sokoban může posunout bednu ve směru svého pohybu za podmínky, že vedle bedny ve směru pohybu není další bedna a ani tam není zeď. Jednotlivé situace znázorňují po řadě obr. 2, obr. 3 a obr. 4, kde Sokoban se ani v jednom případě nemůže posunout doprava.



Obr. 2

Obr. 3



Obr. 4

1.2 Varianty Sokobana

Existuje poměrně velké množství variant Sokobana, které pozměňují jednotlivé aspekty hry, přičemž základ hry je většinou redukován na

„posunování objektů po hrací ploše pomocí uživatelem ovládané postavičky“. Podívejme se tedy na jednotlivé varianty:

- **Alternativní podloží**: Tento typ varianty má stejná pravidla jako Sokoban, liší se pouze v použitém typu políček. Ve standardní hře je bludiště rozděleno na čtvercová políčka. Alternativami jsou například Hexoban, který využívá šestiúhelníkové (hexagonální) políčka, a Trioban, který je založen na rovnostranných trojúhelnících.

- **Alternativy s rozdílným pojetím hlavní postavičky**: Ve variantách *Multiban* a *Interlock* hráč ovládá více postaviček. Vznik varianty *Multiban* je přisuzována Alfredu Pfeifferovi a vznikla až v červnu 2002. Od skladníka jakožto hlavní postavičky se odklonila varianta *Enigma,* ta do hlavní role obsadila černý mramor.

- **Alternativní cíle**: Některé varianty upravují požadavky pro dokončení kola. Například v *Block-o-Mania* jsou bedny s různými barvami a cílem je posunout bedny na cílová políčka tak, aby si barvy odpovídaly. *Sokomind Plus* využívá podobnou myšlenku - s bednami a cílovými políčky, které jsou očíslovány. Ve variantách *Interlock* a *Sokolor* jsou bedny opět různě obarveny, ale cílem je dosáhnout toho, aby bedny s podobnými barvami spolu sousedily. Další variantou je *CyberBox*, kde každé kolo má vyznačené cílové políčko, na které je potřeba se dostat. *Beanstalk* vyžaduje umístění objektů (lopata, nádrž s vodou, semínko fazole, zalévací konev) v daném pořadí na jedno cílové políčko.

- **Varianty s dalšími hracími objekty**: *Sokonex, Xsok, Cyberbox* a *Block-o-Mania* přidávají další objekty k základní variantě hry. Například jámy, teleporty, pohyblivé bloky a jednocestné průchody. V *PocoManovi* jsou bedny nahrazeny poklady, které se při umístění na cílové políčko promění v poklady do dalšího kola.

- **Rozšíření vlastností Sokobana**: Ve variantě *Pukoban* může postavička jak posouvat, tak táhnout bednu.

Kapitola 2

2 Výběr technologií pro aplikaci

Tato kapitola popisuje jednotlivé uvažované technologie pro tvorbu programu, kritéria jejich výběru a vybrané řešení.

2.1 Výběr technologií

Požadavky, kterým se musel výběr technologií podřídit, byly tři:

1. Musí jít o standardní aplikaci pro Windows se standardním grafickým rozhranním a ovládáním.
2. Musí být možné integrovat do aplikace grafickou komponentu pro vykreslení bludiště a animací v něm ve 2D grafice.
3. Pro tvůrce pluginů by mělo být snadné vytvořit nový plugin, nejlépe pouze s pomocí MS Visual Studia.

2.1.1 Samotné XNA

XNA je herní platforma pro herní konzoli Xbox 360 a pro Windows od společnosti Microsoft. XNA příliš nevyhovuje požadavkům popsaným výše, a to z následujících důvodů. Je sice možné vytvořit aplikaci s grafickým rozhranním, ale splnění dalších požadavků je již velmi pracné. V XNA nejsou předimplementovány ani základní ovládací prvky typu tlačítko či rozbalovací menu. Programovat v XNA tedy znamená napsat si veškeré ovládací prvky, které programátor potřebuje, s vzhledem dořešeným do posledního pixelu. Tento fakt podtrhuje i návod [1] na MSDN pro kreslení 3D primitiv jako je například úsečka, což se provádí vykreslením jednotlivých bodů.

Volba XNA by znamenala pro tvůrce pluginů, že by si museli nainstalovat k Microsoft Visual Studiu ještě XNA Game Studio, což je balíček nástrojů pro MS Visual Studio, bez kterého není možné vytvářet hry v XNA, jelikož součásti jsou knihovny pro XNA. Nejde tedy jen o jakési rozšíření MS Visual Studia, které usnadňuje vývoj her. Pro samotné hráče by instalace aplikace vyžadovala navíc instalaci redistribučního balíčku pro XNA, aby aplikace fungovala.

Z důvodů výše popsaných vyplývá, že použití XNA samostatně by pro mou aplikaci znamenalo řešit veškeré problémy dosti nízkoúrovňově a přineslo by to mnoho komplikací.

2.1.2 Samotné Windows Forms

Windows Forms (WinForms) je grafické aplikační rozhranní (API) od společnosti Microsoft, které je dodávané jako standardní součást .NET

Frameworku již od jeho první verze z roku 2002. Velkými výhodami WinForms je nejen to, že tato knihovna je již léty prověřená a na různé problémy jsou známá řešení, ale i fakt, že naučit se pracovat s WinForms je poměrně snadné. Navíc pro WinForms existuje velké množství komponent od třetích stran.

Windows Forms je ideálním kandidátem, co se týče bodů 1) a 3) uvedených v úvodu odstavce 2.1. Potíž je však s vytvořením herní smyčky, což je typicky *while* cyklus, ve kterém probíhají změny v herní logice a změny grafického výstupu. Pro tuto smyčku je zásadní, aby se její obsah zpracovával co nejčastěji a ještě lépe co nejrovnoměrněji. V XNA je pevně daný frame rate (angl. výraz pro frekvenci s jakou se mění jednotlivé snímky na obrazovce) 60 FPS (angl. zkratka pro frames per second), což odpovídá tomu, že změna na obrazovce proběhne jednou za 16,6 milisekund. WinForms žádnou takovou službu nenabízí, je však možné si vytvořit herní smyčku s proměnlivým časem volání. Tři nejjednodušší cesty pro vytvoření hrací smyčky jsou:

- zaregistrování události *Application.Idle,*
- vytvořením nového vlákna pro funkci s *while* cyklem a funkcí *Thread.Sleep()* v těle *while* cyklu a
- použití timeru.

Ještě před popsáním jednotlivých způsobů vytvoření smyčky je důležité poznamenat, jak je možné měnit grafické rozhranní ve formuláři WinForms. Ve WinForms aplikaci je standardně využíváno pouze jedno vlákno pro řízení programu. Toto vlákno jako jediné může měnit vlastnosti grafických komponent, které jsou umístěny na formulářové okno. Pokud máme pouze jedno vlákno, tak tato informace není důležitá, pokud si však vytvoříme další vlákno pro nějaké složitější výpočty, tak jde však již o informaci velmi důležitou, a to proto, že výpočty často potřebujeme na formuláři uživateli prezentovat. Nám však jde o animace a zde je tato informace také důležitá. Na další straně je zdrojový kód v C# pro .NET 2.0, který poslouží jako ukázka, jak lze aktualizovat *text* u grafické komponenty *cntrl*.

```
public class MyForm : Form
{
  delegate void UpdateCntrlTextCallback(Control cntrl, string text);
  public void UpdateCntrlText(Control control, string text)
  {
    if (cntrl.InvokeRequired)
    {
       cntrl.Invoke(
         new UpdateCntrlTextCallback(UpdateCntrlText), cntrl, text
       );
    } else {
      cntrl.Text = text;
    }
  }
}
```

Důležité na tomto zdrojovém kódu je funkce *InvokeRequired*, která zjistí, zda vlákno, které tuto funkci zavolalo je určeno pro grafické úpravy nebo ne. Pokud není, tak pomocí funkce *Invoke* zavoláme přes delegáta funkci *UpdateCntrlText*, ve které se právě nacházíme, ale zavoláme ji přes vlákno, které může dělat grafické úpravy. Volání funkce *Invoke* je synchronní a také relativně pomalé, protože se provádí tak, že se „grafickému“ vláknu pošle zpráva a tam čeká ve frontě na zpracování. Voláním funkce *Invoke* se tedy vyplatí výrazně šetřit.

2.1.2.1 Application.Idle

Nyní zpět k události *Application.Idle*. Tato událost je volána vždy, když aplikace již má zpracovány veškeré zprávy ve své interní smyčce zpráv a nemá nic jiného k zpracování. Je třeba si dávat pozor na to, aby se nevykonávaly časově náročné výpočty v obslužné funkci této události, jinak by se jevila celá aplikace tak, že nereaguje. To samozřejmě platí obecně pro volání funkcí z „grafického“ vlákna aplikace, nicméně tím, že je tato událost volána poměrně často, je nežádoucí „zatuhávání“ aplikace zřetelné. Pokud si to tedy zrekapitulujeme, tak nemáme zaručeno, kdy bude událost zavolána, avšak můžeme rovnou přistupovat ke grafickým komponentám na formuláři. Na mém počítači mělo toto řešení pro aktualizaci formuláře výkon na horní hraně použitelnosti, pokud neběžely na počítači jiné větší úlohy.

2.1.2.2 Thread.Sleep

Druhá avizovaná možnost s vytvořením nového vlákna pro funkci s *while* cyklem a funkcí *Thread.Sleep()* uvnitř může vypadat takto:

```
public void GameLogic()
{
  while (true)
  {
    /* Do some work */
    Thread.Sleep(20); // Number of miliseconds
  }
}
```

Hned prvním problémem tohoto zdrojového kódu je skutečnost, že funkce *Thread.Sleep* nám nezaručuje, že po dvaceti milisekundách se opět bude dále zpracovávat smyčka, zaručuje nám pouze, že to bude minimálně po dvaceti milisekundách. Pokud bychom se pokusili nastavit nějaké velmi malé kvantum času, například pět milisekund, tak nám to stejně nepomůže, protože operační systém sám pracuje s určitými časovými kvanty, která přiděluje aplikacím či vláknům, a po pěti milisekundách nám stejně nepředá řízení. Navíc bychom v takovéto smyčce nemohli přímo upravovat formulářové prvky. Tato herní smyčka je tedy také nevhodná a z mého pohledu je to jedna z nejhorších možných.

2.1.2.3 Timer

Timer je časovač, který pravidelně volá vybranou metodu po uplynutí určité časové periody. Délku periody je možné nastavit. Ve WinForms jsou tři časovače, proberme si možnosti jejich využití pro vytvoření herní smyčky.

2.1.2.3.1 System.Windows.Forms.Timer

Tento časovač je z celé trojice nejméně přesný. Jeho nepřesnost je zapříčiněna tím, že volání obslužné metody časovače se plánuje pomocí zpráv ukládaných do smyčky zpráv aplikace. Zprávy zpracovávané před zprávou časovače jsou typicky obsluhovány různou dobu a tak se zpracování zprávy časovače může výrazně zpozdit. Dle článku [2] je možné určit minimální periodu tohoto časovače na 55 milisekund. To je pro hrací smyčku příliš mnoho a pro vytvoření herní smyčky je tento časovač tedy nevhodný.

2.1.2.3.2 System.Timers.Timer a System.Threading.Timer

S oběma časovači je možno dosáhnout na volání obsluhy časovače po průměrných 20 milisekundách. Obsluha je však volána v jiném vlákně než jaké může upravovat formulářové objekty, tedy bychom museli používat funkci *Invoke*, což nám výkon zpět ubere. Ani tyto časovače tedy pro vytvoření herní smyčky nejsou vhodné.

2.1.2.4 Zhodnocení

Samotné WinForms je pro vytvoření zadané hry nevhodné, protože ani s jednou výše zmiňovanou herní smyčkou nelze dosáhnout kvalitní animace.

2.1.3 Windows Forms s XNA

Dalším řešením, které se nabízelo, bylo, že aplikaci vytvořím pomocí technologií Windows Forms a XNA. Microsoft pro toto spojení technologií vytvořil oficiální ukázku [3], kde XNA je přidáno do okna aplikace jako

komponenta. Nejde však o obvyklé propojení technologií, aplikací využívajících toto řešení jsem nalezl velmi málo.

Výhodou tohoto řešení by bylo snadné vytváření uživatelského prostředí ve WinForms a grafické možnosti XNA.

Snahou o zprovoznění této varianty jsem strávil dohromady několik týdnů. Hlavním problémem bylo vyladit animace pohybů, jelikož se zdálo, že využití *Application.Idle* jakožto herní smyčky bude mít dostatečný výkon, což se ukázalo, že není pravda. Problém částečně zmizel po přechodu z Windows Vista na Windows 7, které mají předinstalovaný .NET Framework 3.5 SP1. Novější .NET spolu s vylepšeními Windows 7 oproti Windows Vista sice problém zmírnily, avšak stále by to znamenalo, že hra by byla nekvalitní ve Windows Vista a tedy jsem od řešení WinForms + XNA upustil.

2.1.4 Windows Presentation Foundation (WPF)

Po neúspěšném pokusu o vytvoření aplikace ve WinForms v kombinaci s XNA mi nezbývalo než přejít k WPF.

WPF je nová technologie Microsoftu, která dovoluje vytvářet grafické aplikace pro MS Windows. Od WinForms se WPF liší v mnoha ohledech, především je možné nativně používat návrhové vzory pro oddělení grafické stránky programu a aplikační logiky. Ve WinForms je nutné si navrhnout takovou architekturu programu svépomocí, to je častým zdrojem problémů, protože dokud je program jednoduchý, tak to programátor nepovažuje za důležité, program se však časem rozroste a najednou přestane být možné testovat aplikaci po částech. Do této pasti se mi s mým programem ve WinForms podařilo také spadnout a nakonec jsem byl rád, když přepisování do WPF mi umožnilo navrhnout novou architekturu programu.

Díky tomu, že WPF je určeno pro vytváření aplikací s bohatým uživatelským prostředím, je možné v této technologii naprogramovat i hru splňující požadavky z odstavce 2.1. WPF však není primárně technologie pro vytváření her, pro 3D hry by bylo její použití nevhodné. Pro 2D hry je však výkon WPF dostatečný.

Důvod, proč jsem nezačal hru programovat již na začátku ve WPF spočíval v tom, že jsem ve WPF předtím nikdy neprogramoval, to byl samozřejmě méně závažný důvod. Druhým a hlavním důvodem byla značná komplexnost této technologie. Pro ilustraci této komplexnosti pouze dodám, že s WinForms bylo možné po načtení několika článků pracovat již daný den, pro porozumění WPF jsem četl týden vynikající knihu [4] a stále jsem si nebyl příliš jistý, že budu schopný programovat aplikace, tak jak by se ve WPF programovat měly.

2.1.4.1 AvalonDock

Při návrhu aplikace ve WPF jsem chtěl, aby se aplikace sestávala z panelů či oken, které by bylo možné přesouvat, skrývat a opět zobrazovat. WPF takovou komponentu neposkytuje, proto jsem se rozhodl využít knihovnu

s otevřenými zdrojovými kódy od Adolfa Marinucciho zvanou *AvalonDock*, jenž je dostupná na oficiální webové adrese projektu [5].

Tato knihovna byla jediná, kterou jsem nalezl a kterou lze použít zdarma (licenční podmínky jsou velmi benevolentní), je aktivně vyvíjená a funguje pod .NET Frameworkem 3.5 SP1. V mém programu je použita verze 1.3 knihovny, která byla uvolněna nedávno a která opravuje známé problémy, které byly ve verzi 1.2. Jmenovitě opravuje verze 1.3 problém zapamatování pozice panelů při skrytí a jejich následném zobrazení, což se mi nedařilo opravit. Jako nevýhodu této knihovny by se jistě dala označit její dokumentace, která prakticky neexistuje.

Kapitola 3

3 Řešitelé kol

V této kapitole bych se rád zaměřil na způsoby, jaké se nabízely při integraci existujících řešitelů kol pro Sokobana a na problémy, které se při tom vyskytly.

3.1 Úvod

Najít existující řešitele kol je možné pomocí různých dotazů do webových vyhledavačů, řešitelů a obzvláště těch lepších je však jak šafránu. Seznam těch propracovanějších je možné najít na webové stránce projektu Sokoban Project [6] a na webové stránce Jorise Wita [7].

3.2 Vznik SolverLibrary

Při svém hledání jsem také narazil na Sokoban Solver SDK [8] (dále jen SDK) od Jorise Wita. Jde o rozhranní napsané v programovacím jazyce C++, které definuje, jaké funkce má podporovat řešitel kol.

Jmenovitě obsahuje SDK funkce: *Configure*, *GetConstraints*, *GetPluginName*, *IsSupportedOptimization*, *Optimize*, *ShowAbout*, *SolveEx*, *Terminate* a zastaralé funkce *Solve* a *GetSolverName*.

Existuje několik řešitelů, které toto rozhranní implementují a je možné je díky tomu používat, aniž by bylo nutné zaobalovat chování řešitelů (například výstup z konzolového programu) pro každého řešitele zvlášť.

Jelikož dané SDK poskytovalo vše, co je pro řešení kol potřeba, rozhodl jsem se napsat třídu v C#, která umožní komunikovat s DLL knihovnami, které toto SDK implementují.

Třídu jsem nazval *SolverLibrary* a její funkce správně volají funkce v DLL knihovnách implementujících SDK. Tato knihovna provádí tzv. marshalling, v mém případě se tento pojem zúžil na proces pro konverzi mezi objekty manažovaného kódu a nativního kódu. Pro funkce v SDK bylo tedy potřeba najít a nastavit správné volací konvence a odpovídající datové typy v C# pro datové typy C++ uvedené v SDK.

Jelikož SDK obsahuje funkce *Configure* a *About*, které vyžadují *handle* okna, jež bude jejich vlastníkem, bylo potřeba vyřešit, jak tento *handle* získat. V .NETu však pro podobné situace existuje třída *WindowInteropHelper*, kterou je možné použít a *handle* tak získat.

SolverLibrary je navržena tak, aby v případě, že plugin podporuje novou verzi SDK, používal nové funkce, a pokud DLL knihovna používá staré SDK, nejdříve knihovna snaží volat nové funkce a pokud neuspěje, pak až teprve

volá staré API funkce. Jde o dvojici funkcí *SolveEx* a *Solve* a druhou dvojici *GetPluginName* a *GetSolverName*.

Projekt *SolverLibrary*, ve kterém se nachází stejnojmenná třída, musel být bohužel zkompilován s direktivou *unsafe*.

Pokud je tato direktiva zapnutá, pak můžeme ve zdrojovém kódu používat pointery a používat paměť, která není spravována Garbage Collectorem (GC) .NETu). Důsledkem je však potřeba pro zvýšenou opatrnost, podobně jako při programování v C++, z důvodu, že Common Language Runtime (CLR; běhové prostředí .NETu) není schopné při zapnuté direktivě *unsafe* odhalit nekorektní program.

Důvodem nutnosti *unsafe* direktivy je fakt, že struktura *PluginStatus* potřebuje pro správnou funkci pole typu *fixed char* (tj. pole, které není spravováno GC, a také proto, že samotná volání funkcí v DLL knihovně je nutné vložit do *unsafe* kontextu.

3.3 Použitelní řešitelé

Zvažoval jsem použití těchto řešitelů: BoxSearch, PocoSolv, YASS (Yet Another Sokoban Solver), Rolling Stone a Takaken Solver. Ostatní řešitelé (jmenovitě například Sokoban Automatic Solver, Sokoban Puzzle Solver,…), které jsem našel, byly často pouze aplikace pro MS Windows s grafickým rozhranním, které využít nelze, případně měly jiné problémy, které jsou popsané v následujících odstavcích.

3.3.1 BoxSearch

BoxSearch je řešitel od čínského programátora Go yonga [9]. Podporuje SDK, bohužel však funguje pouze, pokud je spuštěn z konzolové aplikace (tedy ne z WPF aplikace či WinForms). Autor mi na žádost poslal část zdrojového kódu, který se daného problému týká, ale problém je, že *pipe* pro komunikaci mezi dynamickou knihovnou a aplikací BoxSearch5.exe se nedokáže vytvořit. Zajímavostí je, že řešitel funguje, pokud se jej pokusím zavolat z mého programu, běžícího v MS Visual Studiu v debugovacím módu, v čemž zřejmě hraje určitou roli hostící proces MS Visual Studia pro debugované aplikace.

Jelikož samotný BoxSearch5.exe má pouze grafické rozhranní, nebylo možné řešitele přidat do mého programu.

3.3.2 YASS

YASS [10] je řešitel Briana Damgaarda napsaný v Delphi. Rovněž podporuje SDK, při jeho přidání se však vyskytl zajímavý problém. Po načtení DLL knihovny došlo k výjimce: *System.ArithmeticException: Overflow or underflow in the arithmetic operation.* Docházelo k ní vždy až v mém

zdrojovém kódu, a to v místech, kde zdrojový kód ovlivňoval grafické stránku aplikace (GUI).

Problém se mi podařilo vyřešit pomocí funkce *_fpreset()*, která restartuje balíček pro výpočty s pohyblivou řádovou čárkou. Tuto funkci se obecně radí používat, pokud voláme DLL funkci, o které nevíme, jestli nepřenastavuje přesnost výpočtu s pohyblivou řádovou čárkou. Autor řešitele však ve svém zdrojovém kódu žádné takové výpočty nemá, takže je otázka, co chybu způsobuje. Na druhé straně však je tato dynamická knihovna použita v jiných programech v C++ a tam žádný problém nenastává. Řešitele se mi tedy podařilo úspěšně přidat do mé aplikace.

3.3.3 PocoSolv, Takaken Solver a Rolling Stone

PocoSolv [11] a Takaken Solver [7] podporují SDK a oba fungují bez problémů, proto byly do aplikace přidány.

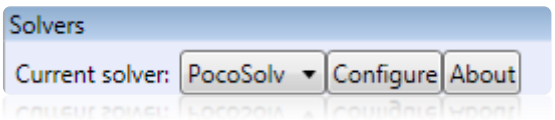
Rolling Stone je projekt University of Alberta [12], jehož zdrojové kódy se mi podařilo zkompilovat pod Cygwinem, což je program, který emuluje unixové prostředí v MS Windows, bohužel však program nemá žádnou nápovědu, jak se s ním vlastně pracuje a tak jsem upustil od snahy napsat pro něj program v C++, který by podporoval SDK a zaobaloval funkce tohoto řešitele.

3.4 SolversManager

SolverLibrary pouze zaobaluje chování DLL knihoven podporujících SDK. Z toho důvodu byla přidána i třída *SolversManager*, jenž dovoluje pracovat s řešiteli již na vyšší úrovni.

SolversManager v rámci inicializace načítá všechny dostupné řešitele ve formě souborů s příponou DLL ze složky %SOKOBAN_DIR%\Solvers. Po spuštění aplikace jsou tedy načteny všichni řešitelé.

Vždy jeden řešitel je nastaven jako vybraný, což koreluje s vybraným řešitelem v nabídce (viz obr. 5) v kartě Solvers v hlavní aplikaci. Tento řešitel je použit pro řešení aktuálního kola při stisknutí tlačítka „Start“ ve stejném panelu. Pokud aktuálně otevřené kolo v hlavní aplikaci není kolo standardní varianty Sokobana, pak řešitel oznámí chybu. V Editoru kol (popsán v kapitole 8) se standardní varianta Sokobana označuje jako „Ordinary“.



Obr. 5. Výběr řešitelů v aplikaci

Jak již bylo zmíněno, *SolversManager* tvoří vyšší vrstvu pro ovládání řešitelů, proto spouštění řešitele při stisku tlačítka „Start“ neprovádí synchronně, ale asynchronně, což nám zajišťuje, že uživatelské rozhranní

aplikace nepřestane reagovat v průběhu řešení kola, které provádí daný řešitel. Pro asynchronní spuštění řešitele byla použita třída *BackgroundWorker*, která je standardní součástí .NETu a je pro takovéto použití přímo určena. *BackgroundWorker* využívá pro svou práci jedno z vláken, které je v „zásobníku“ označovaném jako tzv. *ThreadPool*. Tato vlákna jsou spravována .NETem a tím, že jsou předpřipravená, tak se vyhneme zbytečnému vytváření a destruování vlákna, což má pozitivní vliv na výkon aplikace.

Zvažoval jsem i vytvoření nového vlákna, tedy nové instance třídy *System.Threading.Thread*, což by přineslo možnost zrušení vlákna za běhu pomocí metody *Abort*, toto se však ukázalo jako poměrně nebezpečné, protože to může vést k neuvolnění paměti, či jiných zdrojů v nemanažovaném zdrojovém kódu. Důvod, proč vůbec bylo nutné se touto otázkou zabývat, byl ten, že řešitelé podporující SDK nemusí implementovat metodu *Terminate*, která poskytuje možnost ukončení hledání řešení, pokud uživatel již nechce dále čekat, či z jiného důvodu. Proto se zdála možnost násilného ukončení zpracování vlákna jako dobrá možnost „pro všechny případy“. Kvůli vlastnostem metody *Abort*, jsem od tohoto řešení upustil, přestože snaha o zastavení řešení pracujícího řešitele může trvat od jednotek sekund po desítky v nejhorším případě (jde pouze o experimentálně zjištěné údaje).

Kapitola 4

4 Herní jádro

Tato kapitola popisuje, jakým způsobem funguje základ hry.

4.1 Úvod

Herní jádro je založeno na myšlence diskrétní simulace událostí (DSU). Jde o techniku, která se používá pro modelování chování různých systémů. Model je simulován tak, že se zpracovává posloupnost událostí. Při tomto zpracování se nebere zřetel na reálný čas a místo toho se v každém kroku zpracuje událost s nejnižším časem, tedy ta, která by ve skutečnosti nastala jako první. Čas se tedy mění skokovitě. Typickým příkladem, který se ve spojitosti s DSU uvádí, je modelování chování výtahu nebo například modelování fronty u atrakcí na pouti. Cílem DSU je zjištění, jak se bude model chovat při nastavení různých parametrů, jde o účinnou metodu, která dovoluje model zefektivnit zkoušením změn parametrů (u příkladu s poutí to může být například modelování zisku při přidání další pokladny na lístky s cílem snížit délku fronty).

Herní jádro používá upravenou DSU. Simulace také obsahuje čas (reprezentovaný kladným celým číslem), ten se pravidelně inkrementuje po předem daném časovém intervalu. Události simulace jsou uloženy v kalendáři. V každém kroku simulace se zpracují všechny události kalendáře, které mají čas menší nebo roven aktuálnímu času simulace. Je to hlavně proto, aby žádná událost nemohla být přeskočena a nikdy nezpracována, jelikož herní pluginy mohou přidávat do kalendáře simulace události s libovolnými časy.

Důvodem vybrání upravené DSU byl ten, že umožňuje odprostit se od grafické stránky hry a dovoluje hru reprezentovat jakožto posloupnost událostí. Velkou výhodou tohoto přístupu je i zjednodušení hry dvou hráčů, při té stačí odesílat události přidávané do kalendáře prvního hráče ještě druhému hráči po síti. Není třeba vytvářet nějaký nový protokol pro reprezentaci jednotlivých stavů hry, protože to již události v sobě obsahují. DSU tak řeší oba typy her – pro jednoho hráče i dva hráče – najednou.

4.2 Implementace

Implementace herního jádra zahrnuje implementaci událostí, implementaci kalendáře pro události a vytvoření herní smyčky pro zpracování událostí.

4.2.1 Události

Událost v simulaci představuje struktura *Event*, která obsahuje údaje:

- **when** (kladné celé číslo) – určuje simulační čas, kdy má být událost zpracována,
- **who** (objekt implementující rozhranní *IGamePlugin*, popsané v kapitole 5) – určuje, pro koho je událost určena; speciální hodnota *null* určuje, že událost není určena žádnému hernímu objektu a zpracovat ji může pouze herní jádro,
- **what** (datový typ *EventType*) – určuje typ události, který má být proveden; typy událostí jsou například *goLeft, goRight, hitTheWall, customEvent01,* … Herní objekty (případně herní jádro) pak mají za úkol na tyto zprávy reagovat. Podrobněji je obsluha událostí popsaná v kapitole 5.

Takto navržené události plně postačují potřebám, které se vyskytly při programování herních objektů. Určitou nevýhodou implementace je omezený počet typů událostí. Datový typ *EventType* obsahuje základní události, které se vyskytují běžně (*goLeft, goRight, wentLeft, wentRight, hitTheWall*,…) a dále ještě události, které nemají význam a význam jim dodá až autor herního objektu, jde o události *customEvent01, customEvent02*, …, *customEvent10*. Samozřejmě by se mohlo stát, že autor herního objektu bude potřebovat více jak deset typů událostí, pokud by byl herní objekt dostatečně složitý. Pokud by je potřeboval, pak může využít ještě běžné události jako například *hitTheWall* a mohl by jim přiřadit jiný význam, pokud je nevyužívá. Jsem však zastánce toho přístupu, že pokud by byla nějaký typ události potřeba v mnoha herních objektech, pak by měl být přidán do samotné aplikace a měla by být vydána nová verze aplikace.

Pro typy událostí mohl být zvolen datový typ *string* (řetězec), to by však typicky vedlo k přenášení více dat po síti a vedlo by to k nekonzistencím v používaných názvech událostí. Případně by bylo možné využít datový typ *int* (celá čísla s rozsahem od -2147483648 do 2147483647), toto řešení, ačkoli by problém vyřešilo, jsem nepoužil z toho důvodu, že zdrojový kód herních objektů by nemusel zůstat čitelný, čemuž jsem se chtěl vyhnout odstraněním těchto tzv. magických konstant.

4.2.2 Kalendář

Kalendář simulace je implementován jakožto třída *Calendar*. Do kalendáře je možné přidávat události pomocí metody *AddEvent* a vybírat z kalendáře událost s nejnižším časem pomocí metody *First*, pokud je takových událostí více, pak vrátí tu, která byla do kalendáře přidána nejdříve. *Calendar* představuje svou povahou prioritní frontu. V .NETu není tato datová struktura předimplementovaná. Testoval jsem reálný výkon implementací ([13] a[13]) prioritních front oproti obyčejné struktuře *List* v mé aplikaci a přiklonil jsem se k použití *Listu*, jelikož na testovacích kolech vykazoval větší rychlost. Výsledek si vysvětluji tak, že událostí není v kalendáři typicky mnoho, jedná

se maximálně o desítky událostí v každém kroku simulace, a také tím, že události v kalendáři dlouho nezůstávají, typicky se událost přidá a je zpracována do jedné sekundy.

4.2.3 Herní smyčka

Pro vytvoření herní smyčky jsem použil událost *CompositionTarget.Rendering*, což je událost ve WPF, která je určená pro vytváření animací po jednotlivých snímcích. Počet volání této funkce záleží na vytížení počítače, podobně jako u *Application.Idle* ve *WinForms*. Při velkém vytížení se tedy animace degraduje. S *CompositionTarget.Rendering* jsem byl schopen dosáhnout 60 snímků za sekundu, což je pro animaci velmi dobré. Zkoušet nějaký typ implementace herní smyčky z odstavce 2.1.2 tedy bylo zbytečné, jelikož již z jejich podstaty plyne, že by byly pomalejší.

Herní smyčka je složena z volání:

- Simulace – voláno po uplynutí 30 milisekund od posledního volání
  - Inkrementuj simulační čas.
  - Zpracuj události v kalendáři simulace, které mají simulační čas menší či roven aktuálnímu simulačnímu času.
- Animace
  - Zavolej metodu *Draw* pro všechny herní objekty na hrací desce.

Nevýhoda této implementace je ta, že pokud nějaký herní objekt se zacyklí, pak dojde k zatuhnutí celé aplikace. Jde o daň za rychlé volání vykreslování herních objektů. Herní objekt by však měl být před distribuováním veřejnosti důkladně otestován a tyto problémy by tak nastávat neměly. Tvůrce herních objektů může využít předpřipravené třídy *DebuggerIX* pro výpis ladících informací, který značně zjednoduší hledání chyb. Ladění pomocí této třídy je podrobněji popsáno v odstavci 12.1.

Kapitola 5

5 Herní objekty (pluginy)

Tato kapitola pojednává o principech fungování herních objektů, o vytváření nových objektů dalšími programátory a o možné kooperaci pluginů při hře.

5.1 Úvod

Rozšíření aplikací jsou poměrně častou záležitostí, přesto při hledání možností, jak je skutečně implementovat, jsem narazil jen na pár zajímavých článků. Je nutné poznamenat, že jsem nepotřeboval rozšíření v pravém slova smyslu, potřeboval jsem pouze rozšíření pro herní systém v mé aplikaci. Přesto jsem však hledal nejdříve mezi systémy pro rozšíření celé aplikace jednoduše proto, že by teoreticky mohlo být možné aplikovat stejné postupy pro mé potřeby. Základním požadavkem je tedy samostatně kompilovatelný zdrojový kód, který bude tvořit samotný herní objekt a který půjde dynamicky načíst a používat hlavní aplikací. Dalším požadavkem je co nejvyšší rychlost komunikace mezi aplikací a pluginem, jelikož plugin se musí umět sám vykreslit a obsluhovat události, které jsou mu adresovány z herního jádra.

5.2 Uvažované existující možnosti

Prostředí .NET obsahuje třídu *System.Addin*, což je třída, která dané požadavky splňuje, bohužel jak ukazuje článek [14], je to i systém, se kterým se poměrně obtížně pracuje a pro mé potřeby byl nevhodný zejména z toho důvodu, že by bylo pak obtížné nalézt tvůrce herních objektů, kteří by byli ochotní herní objekty programovat.

Dalším systémem pro práci s rozšířeními je *Managed Extensibility Framework* (MEF). Tento framework však byl veřejnosti představen až po dokončení vlastního systému pro rozšíření (popsáno dále), navíc MEF je primárně určen pro .NET 4 a má aplikace pracuje v .NET 3.5 SP1. Přechod na .NET 4 byl sice možný, jednalo se však a stále se jedná o novou technologii, která nemusí být ještě stoprocentně vyladěná.

5.3 Vybrané řešení

Jelikož mi žádný výše zmíněný systém nevyhovoval, rozhodl jsem se vytvořit si vlastní pluginový systém, který by byl šitý na míru potřebám mé aplikace.

5.3.1 Počet aplikačních domén

Důležitým rozhodnutím bylo, zda umístit pluginy do vlastních aplikačních domén či ne.

Aplikační domény jsou v .NETu jednotky obsažené v procesu, které se používají k oddělení částí aplikace nebo také ke spuštění jedné aplikace vícekrát v rámci daného procesu. Aplikační domény mají oddělené vlastní zdroje a komunikace mezi nimi je možná pomocí tzv. marshallingu. V praxi to znamená, že chyby (např. výjimky) v jedné aplikační doméně nemůžou ovlivnit ostatní aplikační domény v procesu. Ucelené povídání o aplikačních doménách je dostupné v příručce pro programátora na MSDN [15].

Pokud by tedy pluginy běžely v samostatných aplikačních doménách, nebyl by problém s tím, že by pluginy mohly způsobit pád celé aplikace. Na druhou stranu nevýhodou je výrazně pomalejší rychlost komunikace mezi aplikačními doménami oproti standardnímu volání funkce v rámci jedné aplikační domény.

Ve snaze zachovat rychlost aplikace a nevytvářet zbytečně úzké hrdlo jsem se rozhodl pro použití pouze jedné aplikační domény, tedy do aplikační domény aplikace se ještě dynamicky načítají pluginy.

Proti chybám v pluginech je možné se částečně bránit tak, že veškerou komunikaci s pluginem uzavřeme do try – catch bloků, aby se zabránilo pádu aplikace při vyhození výjimky v pluginu. Jde o řešení, kterým neztrácíme výkon, bohužel nás chrání jen před částí možných problémů (například problému zacyklení se takto nevyhneme).

Při reálném provozování aplikace se pak můžeme částečně bránit proti různým problémům s pluginy tak, že na webu budou zveřejňovány pouze ty pluginy, které budou poskytnuty zároveň se zdrojovým kódem, aby bylo možné ověřit jejich chování. Případně zveřejňovat pouze pluginy, které budou nejdříve ozkoušeny komunitou testerů. Toto samozřejmě není programátorské řešení problému, ovšem může značně přispět pro příznivé vnímání celé aplikace koncovými hráči.

5.3.2 Filozofie pluginů

Nejprve je třeba zdůraznit, že veškeré herní objekty jsou vytvořené formou pluginů, není tedy žádný herní objekt, který by měl speciální výsady. Základní pluginy *Sokoban, Monster, Box, Aim* a *Wall*, které jsem programoval, se řídí přesně tím, co je popsáno níže. Výhodou tohoto přístupu je fakt, že je možné i základní pluginy nahradit vlastními a změnit chování podle požadavků. Takovými požadavky by mohly být například změny textur jednotlivých herních objektů nebo třeba vypnutí animací pro hraní na slabších strojích.

Pluginy jsou v mém systému samostatně kompilovatelné dynamicky linkované knihovny, které jsou načítány do hlavní aplikace. Při načítání kola z XML souboru jsou pak vytvářeny instance načtených pluginů. Obsahuje-li tedy hrací kolo zdi kolem celého bludiště, je vytvořeno tolik instancí pluginu *Wall*, kolik je těchto zdí kolem bludiště.

Aplikace komunikuje s pluginem na základě rozhranní *IGamePlugin* (k nahlédnutí v odstavci 5.3.3.5) či na základě rozhranní *IGameVariant* (k nahlédnutí v odstavci 5.3.3.6) a plugin může komunikovat s aplikací pomocí rozhranní *IPluginParent* (k nahlédnutí v odstavci 5.3.3.7). Všechna volání v herním jádře jsou prováděna synchronně, je to proto, že komunikace je považována aplikací za rychlou. Plugin by tedy neměl zbytečně zpomalovat celou hru náročnými operacemi. Ostatně pokud potřebuje plugin vykonávat nějaké časově náročné operace, může tyto operace přesunout do nového vlákna.

Jsou tedy dva druhy pluginů, první druh musí implementovat rozhranní *IGamePlugin* (dále označováno pouze jako „plugin“) a druhý typ musí implementovat rozhranní *IGameVariant* (dále označovány jako „pluginy variant“).

Plugin implementující rozhranní *IGamePlugin* jednak definuje svá inicializační data prostřednictvím XML schémata, dále svůj vzhled a své chování při hře. Použití XML v těchto pluginech je popsáno v odstavci 5.4. Nyní se podívejme podrobněji na vzhled pluginu.

5.3.2.1 Vzhled pluginů

Díky rozhranní *IGamePlugin* musí každý plugin obsahovat instanci třídy *UIElement* (pojmenováno též *UIElement,* viz rozhranní *IGamePlugin* v odstavci 5.3.3.5), což je třída, která se používá ve WPF pro všechny komponenty, které je možné umístit na formulář. Tvůrci herních objektů si mohou vybrat z mnoha možností, jak bude jejich výsledný plugin vypadat, a to proto, že mnoho tříd ve WPF od třídy *UIElement* dědí a tedy je možné tyto potomky pro definování vzhledu použít. Daný *UIElement* je přidán jako potomek *Canvasu*, což je grafická komponenta, která je použita pro zobrazení hrací desky v aplikaci. Toto nastane po vytvoření všech instancí pluginů pro načítané kolo. *UIElement* je vhodné přiřadit v metodě *Load*.

Rozhranní *IGamePlugin* nutí pluginy k implementaci metody *Draw*. Tato metoda je volána hlavní aplikací, kdykoli je to možné. Je však nutné si dát pozor na to, že v aplikaci lze hrát pouze jednu hru naráz (tu, která je vidět na obrazovce), přestože je možné mít otevřeno více her v aplikaci. Proto je třeba počítat s tím, že volání metody *Draw* ustane, jestliže hráč přejde k jinému kolu, které má v záložkách, či si spustí nové.

Není dán pevně čas, jak často volání metody *Draw* budou opakována, ale je možné počítat s desítkami milisekund (na mém stroji je to 20 až 30 ms, v případě větší zátěže procesoru se může vyskytnout delší prodleva, typicky je však výkon pro animaci dostatečný). Pokud se plugin neanimuje, pak nemusí v této metodě provádět téměř nic, pokud ano, pak sem může autor vložit příslušný kód, který animaci provede. Je třeba však počítat s tím, že hrací deska nemá pevně danou velikost políček, je tedy nutné, aby se plugin uměl této situaci přispůsobit. Velikost políček není pevná z toho důvodu, že hráč si

může rozšířením hracího okna zvětšit i hrací desku a této situaci se musí umět herní objekty přizpůsobit, proto tedy i herní objekt, který se neanimuje, se musí příslušně zvětšovat a zmenšovat.

5.3.2.2 Chování pluginů

Jak již bylo popsáno v kapitole 4 o herním jádře, základem hry je upravená diskrétní simulace se zprávami. Aplikace tedy prohlíží kalendář událostí a zprávy rozesílá pluginům, kterých se zpráva týká. Zpráva je předána pluginu aplikací pomocí metody *ProcessEvent* spolu s aktuálním časem.

Zpráva je poslána jakožto struktura:

```
public struct Event
{
    // When to process the event
    public Int64 when;
    // Which object should process the event
    public IGamePlugin who;

    /// Type of event
    public EventType what;
}
```

Aktuální čas je posílán z toho důvodu, že plugin při vkládání nových událostí nemá omezeno, jaký čas určí pro zpracování událostí. Herní jádro zpracuje i události s časem, který je nižší než aktuální. Toto se dá použít, pokud by bylo třeba provést okamžitě nějaké události v daném pořadí, jelikož herní jádro zpracovává v každém kroku všechny události, které mají časy menší rovné aktuálnímu času.

Událost obsahuje položku *what*, která určuje typ události, který se má provést. Seznam všech dostupných událostí je možné nalézt ve zdrojovém kódu, zde jen uvedu, že *EventType* je *enum* (jazykový konstrukt pro pojmenovaní číselných hodnot sloužící pro udržení čitelnosti programu), který vyjmenovává události. Například jde o události *goLeft, goRight, hitToTheWall, customEvent01, customEvent02,* … Události *customEvent* může autor pluginu použít pro libovolný účel, u ostatních by se měl snažit je používat pro to, co označují, přestože to není vyžadováno.

Pokud je plugin schopen zprávu zpracovat, pak ji zpracuje a musí vrátit jako návratovou hodnotu hodnotu *True* v metodě *ProcessEvent*, pokud zprávu zpracovat nedokáže, pak by měl vrátit *False* a o zpracování se pokusí herní jádro, pokud ani to zprávu nedokáže zpracovat, pak se zpráva odstraní a pokračuje se ve zpracovávání zbývajících událostí.

Ještě si popišme zpracování události na příkladu. Zpracovat událost znamená jednoduše to, že provedeme akci spojenou s přijatou událostí. Pro ukázku zpracování je možné se podívat do pluginu *Sokoban* a podívat se na jeho obsluhu metody *ProcessEvent*. Pokud tato metoda zpracovává událost, která nařizuje hernímu objektu, že se má pohnout doprava (typ události je označován jako *goRight*), tak se pozmění hodnota proměnné *PosX*, čímž se

změní logická polohu herního objeku, a jelikož pohyb mezi dvěma políčky má být animován, tak po vyhrazenou dobu (definovanou proměnnou *speed* v pluginu jakožto periodu simulačního času pro pohyb) se bude v metodě *Draw* s pluginem drobně pohybovat po hrací desce, dokud nebude herní objekt na správném políčku.

5.3.3 Načítání pluginů a pluginů variant do aplikace

Pluginy i pluginy variant, jakožto dynamicky linkované knihovny (DLL), jsou do aplikační domény hlavní aplikace načítané pomocí třídy *System.Reflection.Assembly* a metody *LoadFrom*. Pro úspěšné načtení pluginu (jakožto typu, ne instance) je potřeba, aby plugin implementoval rozhranní *Sokoban.Model.PluginInterface.IGamePlugin* (k nahlédnutí v odstavci 5.3.3.5) či rozhranní *Sokoban.Model.PluginInterface.IGameVariant* (k nahlédnutí v odstavci 5.3.3.6).

Načítání řeší v mém programu třída *PluginService*. Každé okno s hrou v hlavní aplikaci obsahuje jednu instanci této třídy. Protože načítat DLL knihovny s pluginy stačí pouze jednou, tak tato třída obsahuje statické položky, kam se načtou pluginy (opět typy, ne instance) a jsou tak dostupné ze všech instancí třídy *PluginService*. Instance třídy *PluginService* pak vytváří instance z načtených pluginů ve statických položkách.

PluginService obsahuje základní metody pro základní práci s pluginy a pluginy vaiant:

- *void LoadAssembly(string fileName);* Tato metoda načte dynamicky linkovanou knihovnu, ve které je třída implementující rozhranní *Sokoban.Model.PluginInterface.IGamePlugin* či *Sokoban.Model. PluginInterface.IGameVariant* do aplikační domény hlavní aplikace. Metoda testuje zda skutečně obsahuje DLL knihovna obsahuje třídu implementující jedno či druhé rozhranní. Děje se tak s pomocí *Reflection*, což je třída .NETu, která dokáže za běhu aplikace zjišťovat informace o třídách, objektech, obsahu assembly apod.
- *string GetPluginSchema(string fileName);* Každý plugin musí specifikovat své XML schéma pro inicializační data. Podrobněji je tato problematika popsána v odstavci 5.4.
- *IGamePlugin RunPlugin(string pluginName);* Metoda vytvoří a vrátí novou instanci daného pluginu. Pomocí funkce *LoadFrom* jsme totiž načetli jen danou DLL knihovnu, pro její „použití“ je však potřeba vytvořit instance daného pluginu, což právě dělá tato funkce.
- *IGameVariant RunVariant(string variantName);* Obdoba metody *RunPlugin* pro pluginy variant.

- *void CloseAllPlugins();* Metoda volá metodu *Unload* pro všechny instance běžících pluginů a instance pluginů variant, aby se tak mohly odstranit případné nemanažované zdroje pluginů.

5.3.3.1 Programování pluginů

Zdrojový kód pluginu může být poměrně krátký, jak je možné se přesvědčit ve zdrojovém kódu pluginu *Sokoban*. Je to proto, že třída *Sokoban* dědí od předpřipravené třídy *MovableEssentials*, která obsahuje základ pro plugin, který se bude pohybovat, obsahuje však i finesy jako je například klávesový buffer pro pluginy ovládané uživatelem, podporu pro odtlačení jiného objektu v závislosti na jeho „hmotnosti“ (v kódu proměnná *obstructionLevel*) a jiné.

Pro zkompilování pluginu je nutné přidat reference na projekty *Lib* a *PluginInterface*. Případně použít jako reference soubory Lib.dll a PluginInterface.dll.

Pro programování pluginů je také podstatný typ pluginu, jak je uvedeno v následujícím odstavci.

5.3.3.2 Typy pluginů

Pluginy se logicky dělí na několik kategorií. Hlavním dělícím znakem je, zda se plugin může pohybovat či ne. Dalším je, zda plugin může být ovládán hráčem a posledním je zda jde o plugin, který je „deskou“ (typickým příkladem je cílové políčko, po kterém se může pohybovat Sokoban či bedna ve standardní variantě) nebo objektem (příkladem je Sokoban, zeď a bedna ve standardní variantě) na hrací desce.

Pro označení typu se používá implementace jednoho či více rozhranní pro určení typu. Jde o rozhranní *IFixedElement, IFixedTail, IMovableElement* a *IControllableByUserInput*. Plugin však musí být buď „Element“ (plugin je objektem) nebo „Tile“ (plugin je „deska“) a dále buď „Movable“ (plugin se může pohybovat) nebo „Fixed“ (plugin se nepohybuje). Implementace rozhranní *IControllableByUserInput* pluginem pak zajišťuje, že aplikace bude pluginu předávat informace o všech stisknutí a uvolnění kláves klávesnice hráčem.

Nastavení typu pluginu pomocí implementace rozhranní je velmi důležité pro výkon aplikace. Pluginy, které jsou nepohyblivé, jsou ukládány do matic, aby k nim byl rychlý přístup, kdežto pohyblivé pluginy jsou ukládány do polí a přístup k nim trvá déle.

5.3.3.3 Programování pluginů variant

Pluginy variant určují variantu hry, která se bude hrát. Tyto pluginy shromažďují informace o průběhu hry od ostatních pluginů (přístupné přes *IPluginParent.AllPlugins*), případně jim je ostatní pluginy posílají pomocí metody *IPluginParent.CheckRound*. Dále pluginy variant určují, kdy hra končí a s jakým stavem, tj. zda hráč vyhrál či prohrál.

V každém kole je pouze jeden plugin varianty (PV), ten je určený hodnotou v tagu <*Variant*> v XML zápisu kola. Výsadou PV je fakt, že po každém zpracování všech událostí v kalendáři v jednom kroku simulace se zavolá metoda *CheckRound* daného PV, kde se předpokládá, že PV zkontroluje, zda hra neskončila, pokud skončila, pak může PV přidat simulační událost *gameWon* či *gameLost*, které dokáže herní jádro zpracovat a postarat se o vyrozumění uživatele.

PV může dále shromažďovat informace o pohybech pluginů implementujících rozhranní *IMovableElement* pomocí události *ElementMoved*.

PV může komunikovat s pluginy nad rámec toho, co poskytují jednotlivá rozhranní pomocí metody *IGamePlugin.MessageReceived*. Pluginy mohou odpovídat pomocí metody *IPluginParent.GameVariant.CheckRound*.

5.3.3.4 Zvuky

Pluginy mohou samozřejmě používat i zvuky. Pokud plugin chce, aby zvuk byl spravován aplikací, pak je možné použít funkci *RegisterMediaElement* v metodě *Load*. *MediaElement* je standardní součást WPF, tedy se zvuky nebude mít autor pluginu problémy.

Jediný problém je ten, že třída *MediaElement* nemůže jako zdroj dat použít „zdroje“ (angl. resources; jde o způsob ukládání různých typů dat přímo do assembly, tedy do DLL knihovny či EXE souboru, který zjednodušuje správu dat, které assembly potřebuje) v assembly. Je možné načíst pouze zvukový soubor ze souborového systému.

Důvod pro registraci instancí třídy *MediaElement* je ten, že aplikace pak může ovládat zvuk centrálně a pokud si hráč v nastavení vypne zvuk, pak je možné všechny zvuky ztlumit centrálně.

Autor nemusí samozřejmě registrovat svůj *MediaElement* a může použít *MediaPlayer*, což je také standardní součást WPF a může přehrávat zvuky bez omezení.

Pointa tohoto řešení spočívá v tom, že autor může využít registrace (což je preferované řešení), ale nemusí.

5.3.3.5 IGamePlugin rozhranní

```
interface IGamePlugin
{
    // Plugin identification
    string Name { get; }
    // Plugin description
    string Description { get; }
    // Your name
    string Author { get; }
    // Version of your plugin; e.g. 1.00;
    string Version { get; }
    // For what version of SoTh application was the plugin created;
    // e.g. 1.00;
    string CreatedForHostVersion { get; }
    // Appearance of your plugin; it may be image or a geometric
```

```
    // shape etc. Whatever object that inherits from UIElement
    UIElement UIElement { get; set; }
    // Returns XML schema for what is should be content of tag
    // <YourPluginName></YourPluginName> in round specification
    string XmlSchema { get; }

    // Your initialization goes here; appPath param is without
    // trailing backslash
    void Load(string appPath);
    // In case that you use some unmanaged code and you need to
    // correctly destruct an object, ...
    void Unload();
    // Plugin has to draw itself on gamedesk
    void Draw(Canvas canvas, double squareSize, Int64 time,
      double phase);
    // Returns true if initialization was successful
    // settings is content of tag <YourPluginName></YourPluginName>
    // in round specification
    bool ProcessXmlInitialization(string gameVariant, int mazeWidth,
      int mazeHeight, XmlNode settings);
    // Return false if plugin is not able to process the event,
    //host will take care of the message
    bool ProcessEvent(Int64 time, Event e);
    // For cross-plugin communication
    void MessageReceived(string messageType, object message,
      IGamePlugin sender);
    int ID { get; set; }
}
```

5.3.3.6 IGameVariant rozhranní

```
interface IGameVariant
{
    // Plugin identification
    string Name { get; }
    // Plugin description
    string Description { get; }
    // Your name
    string Author { get; }
    // Version of your plugin; e.g. 1.00;
    string Version { get; }
    // For what version of SoTh application was the plugin created;
    string CreatedForHostVersion { get; }
    // Return the solution for given round; used for standard Sokoban
    // variant
    string GetSolution();

    // Your initialization goes here;
    // appPath param is without trailing backslash
    void Load(string appPath);
    // In case that you use some unmanaged code and you need to
    // correctly destruct an object, ...
    void Unload();

    void CheckRound(Int64 time);
    void CheckRound(Int64 time, string messageType, object data);
}
```

5.3.3.7 IPluginParent rozhranní

```
interface IPluginParent
{
    // For communication between plugins
    void Message(object message, IGamePlugin plugin);

    // Plan an event for game simulation
    void MakePlan(string debug, Int64 when, IGamePlugin who,
      EventType what);
    // Plan an event for game simulation with current simulation time
    void MakeImmediatePlan(string debug, IGamePlugin who,
      EventType what);

    // Gamedesk actions

    // Method finds an object that is on position [x, y] and returns it
    // or it returns null
    IGamePlugin GetObstructionOnPosition(int x, int y);

    // List of all plugins in given round
    List<IGamePlugin> AllPlugins;

    // Process all events of game simulation; provides a speedup for
    // situation when an event need to be processed immediately;
    // The method should be used only in special cases.
    void ProcessAllEvents();
    void ProcessAllEvents(double phase);
    void ProcessAllEvents(bool updateTime, double phase);

    // For signalling of change of the number of steps
    void PropertyChanged(string name);

    // Registering sounds
    void RegisterMediaElement(MediaElement me);
}
```

5.4 Využití XML v pluginech

V Příloze A je uvedeno XML schéma definující ligu (tedy soubor kol), kterou umí zpracovat hlavní aplikace. Toto schéma definuje, že v každém kole musí být seznam herních objektů (značka <*GameObjects*>), který jako své potomky může obsahovat elementy *GameObject*.

Pokud by všichni potomci elementu *GameObjects* se jmenovali *GameObject*, pak by zápis kola byl velmi nepřehledný. Chtěl jsem, aby bylo možné přidávat do *GameObjects* potomky podle jmen herních objektů – tedy například tak, jak je uvedeno v ukázce na další straně.

```
...
<GameObjects>
  <Sokoban>
    <!—Sokoban’s initialization data -->
  </Sokoban>
  <Wall>
    <!-- Initialization data of Wall -->
  </Wall>
</GameObjects>
...
```

Ukázka 1

Na příkladu výše to není vidět, ale rozumným požadavkem je také to, že nebude záležet na pořadí zápisu jednotlivých typů pluginů, čehož jsem chtěl také docílit. Zmíněné požadavky jsou rozumné v tom smyslu, že hráčům umožňují bez problémů číst zápis kol.

XML toto skutečně umožňuje. Dosáhnout požadovaného efektu lze nastavením atributu *type* u elementu *GameObject* a dále nadefinováním typu *GameObjectType*, což je základní typ, který bude určovat vlastnosti každého herního objektu.

```
<!-- Game objects -->
<xs:element name="GameObject" abstract="true" type="GameObjectType" />

<xs:complexType name="GameObjectType">
  <xs:sequence>
    <xs:element name="Version" type="xs:string" minOccurs="0"
                maxOccurs="1" />
    <xs:element name="PosX" type="xs:integer" />
    <xs:element name="PosY" type="xs:integer" />
    <xs:element name="InitializeMessage" type="xs:string" minOccurs="0"
                maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
```

Ukázka 2

Od typu *GameObjectType* pak můžeme vytvářet rozšíření (angl. extension; princip je velmi podobný konceptu dědičnosti v objektovém programování), které můžeme vidět na ukázce XML schématu č. 3, kde se definuje element *Monster*.

Jelikož nyní máme vytvořený element *Monster*, který rozšiřuje element *GameObject*, tak je možné do ukázky č. 1 přidat potomka elementu *GameObjects* se jménem *Monster* a spolu se schématem z ukázky č. 3 a schématem z Přílohy A bude XML validní. Validování je vysvětleno hned v následujícím odstavci.

Validace je proces, při kterém se ověří, že XML dokument (v našem případě popis kola) odpovídá XML schématu, které definuje, jak má XML dokument vypadat. Jde tedy o šablonu pro zápis XML dokumentů.

Validace kola skrývá jednu drobnou potíž. Nevíme, jaké herní objekty jsou v XML kola použity a abychom mohli XML validovat, potřebujeme schémata všech herních objektů použitých v daném kole.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns="http://www.martinvseticka.eu/SoTh"
           targetNamespace="http://www.martinvseticka.eu/SoTh"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">


  <xs:element name="FirstState">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="wait"/>
        <xs:enumeration value="goRight"/>
        <xs:enumeration value="goUp"/>
        <xs:enumeration value="goDown"/>
        <xs:enumeration value="wait"/>
        <xs:enumeration value="pursuit"/>
        <xs:enumeration value="guardRow"/>
        <xs:enumeration value="guardColumn"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

  <xs:complexType name="MonsterType">
    <xs:complexContent>
      <xs:extension base="GameObjectType">
        <xs:sequence>
          <xs:element ref="FirstState" />
        </xs:sequence>
       </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="Monster" substitutionGroup="GameObject"
              type="MonsterType" />

</xs:schema>
```

Ukázka 3

Bylo by sice možné číst XML kola, aniž bychom spoléhali na validitu, ale mně se toto řešení nelíbilo a přišlo mi příliš spekulativní.

Máme tedy jedno hlavní schéma prezentované v Příloze A a dále každý plugin má své vlastní schéma a tato všechna schémata chceme použít pro validaci jednoho kola. C# umí validovat XML dokument podle více XML schémat, v tom problém není a ušetří nám to práci s jejich spojováním. Důsledkem je nutnost načítání všech herních objektů při startu aplikace pro získání jejich XML schémat. Není to ideální, jelikož nějaké pluginy by jinak nemusely být vůbec načteny, ale můžeme zato jednoduše validovat XML zápis kol.

Kapitola 6

6 Síťová hra

Tato kapitola popisuje implementaci a možnosti hry dvou hráčů v hlavní aplikaci.

6.1 Úvod

S rozšiřováním Internetu se stále více aplikací stává síťovými. V mé aplikaci představuje síťová hra způsob, jak ze hry, která je tradičně pro jednoho hráče vytvořit hru pro hráče dva a ozvláštnit ji tak. To samozřejmě platí pouze pro standardní variantu Sokobana, je totiž možné vytvořit pomocí pluginů hru, která se Sokobanovi nepodobá.

Síťová hra se hraje tak, že každý hráč vidí při hře svou hrací desku a zároveň vidí desku soupeře. Při hře vyhrává ten, kdo stihne dříve vyřešit dané kolo. Pokud hráč dané kolo zkazí, může svou hrací desku restartovat a hrát znovu, tím však hráč přichází o čas a druhý získává výhodu. O bodovém hodnocení hry je možné se dočíst v kapitole 9.

6.2 Implementace

Pro síťové spojení je použit protokol *Transmission Control Protocol* (TCP) nad protokolem IP. Tento protokol spojuje dvě komunikující strany a umožňuje mezi nimi posílat data. Důležitými rysy protokolu je zajišťování oprav chyb, které mohou vzniknout při přenosu, a přijímání dat v takovém pořadí, v jakém byly odeslány. Druhým protokolem na výběr byl *User Datagram Protocol* (UDP), který tyto dvě zmiňované vlastnosti nemá a tak by bylo nutné je doimplementovat, to samozřejmě možné je, ale při špatné implementaci by to mohlo vést k větší zátěži sítě, než vyžaduje TCP protokol. Dal jsem přednost robustnosti protokolu TCP, protože jeho použití je jednodušší a protože je to velmi často používaný protokol.

6.2.1 Třídy NetworkServer, NetworkClient a TcpConnection

TcpConnection je třída, která založená na třídě *Socket* z .NETu. Úkolem této třídy je odesílání a přijímání dat pomocí asynchronních metod z třídy *Socket*. Asynchronní odesílání (metoda *BeginSend*) a přijímání (metoda *BeginReceive*) je zvoleno z toho důvodu, že nechceme blokovat čtení kvůli zápisu a opačně, tyto metody totiž nečekají až odešlou či přijmou nějaká data, ale zaregistrují požadavek a až jsou data odeslaná či připravená, tak se zavolá příslušná obsluha, kde na akci reagujeme. Velkou výhodou tohoto přístupu je i to, že máme-li vlákno, které řeší akce spojené se síťovou činností, pak toto

vlákno může být snadno přerušeno, což by při použití synchronních volání neplatilo.

Třídy *NetworkServer* a *NetworkClient* obalují třídu *Socket* a zajišťují připojení a odpojení pro roli, ve které uživatel je (tedy klient či server). Navíc obě třídy dědí od třídy *TcpConnection*, odesílání a přijímání zpráv tedy bylo naprogramováno pouze jednou.

Po navázání spojení mezi klientem a serverem již není třeba rozlišovat mezi tím, kdo je v jaké roli a proto jsem vytvořil rozhranní *IConnection*, které *NetworkServer* a *NetworkClient* implementují. Komponenta s hrací deskou pak obdrží instanci *IConnection*, se kterou se snadno pracuje.

6.2.1.1 Zprávy

Třída *TcpConnection* řeší způsob, jakým jsou odesílány zprávy týkající se herního protokolu. Řešení spočívá v tom, že pro odeslání jedné zprávy se posílají socketem zprávy dvě – první určuje typ zprávy (jde o enum strukturu *NetworkMessageType*) a délku následující zprávy poslanou přes socket. První zpráva má pevnou délku 8 bajtů. Druhá zpráva pak obsahuje samotná data, což jsou serializované objekty.

Posílání dvou zpráv je častý způsob řešení. Další možností je posílání zpráv pevné délky, což je samozřejmě také možné, jen by se zpráva musela rozdělit na bloky a na straně příjemce opět spojovat v jednu zprávu. Přenášení zpráv by pak bylo lehce složitější, proto jsem vybral první řešení, které mi přijde elegantní.

Pro přenášení zpráv po síti je typicky nutné zprávy při odesílání kódovat a při přijímání dekódovat. Pokud komunikace probíhá mezi dvěma různými platformami, tak je to způsob, jak zajistit, že si obě strany budou rozumět, protože budou obě strany přesně vědět, co který bit či bajt znamenají.

Má aplikace ovšem komunikuje se stejnou aplikací na stejné platformě a se stejnou verzí .NET Frameworku, proto je možné v tomto případě použít tzv. serializaci. Serializace je způsob, jakým lze převést objekty v paměti do posloupnosti bajtů, kterou je pak možné uložit například do souboru anebo, jak je tento proces použit v mé aplikaci, posloupnost bajtů odeslat po síti druhé aplikaci. Druhá aplikace pak může data deserializovat, tedy převést data z posloupnosti bajtů opět do podoby objektů uložených v paměti.

Pokud by se ukázalo, že serializace je nedostatečně rychlá, pak je možné díky externím knihovnám použít rychlejší serializaci než nabízí standardně .NET Framework. Hlavní výhodou je ovšem možnost snadného dodání dalších typů zpráv, což pokud by bylo použito vlastní kódování a dekódování by znamenalo úpravu i tohoto protokolu.

6.2.1.2 Odesílání dat o hře

Jak bylo popsáno v kapitole 4, herní jádro je založené na diskrétní simulaci. Zprávy, které se zpracovávají, je možné velmi jednoduše kopírovat a zároveň je posílat protivníkovi.

Vyskytuje se zde však ještě jeden problém, který je nutné řešit. Přenos dat po síti určitou dobu trvá a je nutné tento čas kompenzovat. Proto se pravidelně zasílá ještě informace o čase odeslání a o aktuálním času diskrétní simulace druhému hráči. Pomocí těchto informací je možné odhadnout zpoždění, se kterým se má zobrazovat aktualizace na soupeřově desce tak, aby nedocházelo k vyprázdnění bufferu a pak rychlému dohánění času, například v podobě zrychlených pohybů.

Odhad zpoždění přenosu dat po síti je nastaven na hodnotu $D = 300$ milisekund, tedy na deset kroků simulace hry, jelikož čas diskrétní simulace se v herním jádru zvyšuje po 30 milisekundách. Jde o čas, který nebude spoluhráči vnímán jako velké zpoždění a zároveň je pro dobré internetové připojení snadno dosažitelný.

Pokud přenesená data o událostech obsahují simulační čas T, který se liší od aktuálního času simulačního času na protivníkově desce P o méně než deset událostí, tak nastavíme D na rozdíl těchto časů vynásobený 30 milisekundami a získáme čas, které nám simuluje zpoždění přenosu dat.

Čas D se zmenšuje o 10 milisekund pokaždé, když vypočtené D je menší než původní D, nejnižší hodnota D je však 300, jak bylo uvedeno výše. Pokud je nové D větší než původní D, pak si nové D zapamatujeme. Tímto docílíme pomalého zmenšování zpoždění, pokud se připojení zlepší.

Kapitola 7

7 Výběr technologií pro web

Tato kapitola popisuje, jaká kritéria byla zvolena pro výběr technologií pro webovou část mé práce.

7.1 Úvod

Zadání práce jasně říká, že webová část mé práce bude naprogramována ve skriptovacím jazyku PHP s pomocí databáze MySQL. Popišme si, o jaké nástroje se jedná.

Jazyk PHP (rekurzivní zkratka odvozena ze slov „PHP Hypertext Preprocessor“), jak již bylo zmíněno, je skriptovací jazyk, který se používá hlavně pro vytváření dynamických webových stránek. Pro jeho používání je nutné nainstalovat na webový server PHP modul (distribuovaný zdarma a navíc s otevřeným zdrojovým kódem), nejčastěji jako modul *mod_php* v tandemu s webovým serverem *Apache*. Dynamická webová stránka je soubor, typicky s příponou „php“, jež je před odesláním klientovi webovým serverem zpracován PHP interpretem (modulem), který vrátí výslednou HTML stránku. Zpracování tedy probíhá na serverové straně a klientovi stačí webový prohlížeč pro zobrazení výsledku od PHP interpreta.

O popularitě PHP vypovídá fakt, že PHP verze 4 je nainstalováno na přibližně dvaceti procentech všech internetových domén [16].

MySQL je relační databáze s otevřeným zdrojovým kódem, jenž je distribuovaná zdarma. MySQL do velké míry splňuje ANSI/ISO SQL standard, rozdíly je možné najít v oficiální dokumentaci [17].

7.2 Možnosti nasazení

Obecně hlavní předností webových projektů založených na PHP a MySQL je to, že nemusíme mít vlastní webový server a stačí si zakoupit téměř libovolný webový hosting a můžeme si být jistí, že PHP a MySQL budou součástí hostingového programu. V současné době jsou hostingové programy natolik široké, že jestliže základní hostingové programy nedostačují našim požadavkům, je možné si pronajmout dedikovaný server, což je server, na kterém běží několik málo webových projektů (nižší agregace klientů než u obyčejného hostingové programu) a možnosti nastavení jsou zde větší.

Má webová aplikace je umístěna na webovém hostingu společnosti Pipni.cz s nejlevnějším placeným hostingovým programem. Hardwarové nároky samotné webové aplikace jsou dle mých zkušeností z předchozích projektů minimální. Jak vytěžuje aplikace webový server, by však bylo potřeba ověřit

experimentálně. Podle mého názoru by současný hardware a hostingový program stačil minimálně pro potřeby 200 hráčů hrajících současně.

7.3 Implementační možnosti

Při programování v PHP jsme hned na začátku postaveni před volbu, zda použít či nepoužít nějaký framework. Jde o volbu zásadní.

7.3.1 Čisté PHP

Použití čistého PHP má velkou výhodu v tom, že je možné kontrolovat množství zabrané paměti a lze snadno optimalizovat kód pro vyšší rychlost. Tyto snahy však často končí vytvořením nového PHP frameworku a tak se znovu a znovu vynalézají věci, které jsou v jiných frameworcích hotové. Problémem u PHP je nedostatečná podpora práce s HTML a zvláště validace formulářů. Není zabudována ani žádná podpora pro HTML šablony pro zobrazení výstupu a tak častou začátečnickou chybou je míchání aplikační logiky a grafického výstupu.

Častým nešvarem u aplikací v čistém PHP je také nedostatečné zabezpečení. Nejznámější je nyní zřejmě SQL injection, což je bezpečnostní zranitelnost (často s fatálními důsledky) umožňující pozměnění dotazu na databázi.

Problémy s bezpečností je z velké části zapříčiněn tím, že vstupní parametry skriptu se opomenou správně ošetřit. Opomenutí nebo i záměrné neošetření kódu je důsledkem stále dokola se opakujícího kódu, což je problém i u validace formulářů.

7.3.2 PHP frameworky

První možností, která se nabízela, byl můj vlastní „framework“, který jsem začal tvořit ještě v době, kdy se příliš nepoužívaly objekty v PHP (tj. PHP 3). Přestože tento framework pracoval spolehlivě, jeho modernizace by zabrala příliš mnoho času a výsledkem by byl pouze další v řadě existujících frameworků bez dokumentace.

Jelikož jsem na počátku žádné PHP frameworky neznal, tak výběr byl široký. Dle srovnání [18] mezi nejznámější frameworky patří: Akelos, CodeIgniter, CakePHP, Jelix, Kohana, Fuse, Prado, Qcodo a Zend Framework. V České republice se pak těší oblibě Nette Framework od Davida Grudla jakožto zřejmě jediný český PHP framework, který je v pokročilém stádiu vývoje a je veřejně dostupný.

Při výběru jsem se snažil najít takový framework, který je rychlý, nezabírá příliš mnoho paměti a má dobrou dokumentaci. Ze srovnání PHP frameworků [18] a [19] lze vyvodit, že Zend Framework je poměrně pomalý. Je sice možné použít eAccelerator, což je optimalizér, který ukládá do cache-paměti zkompilované PHP soubory, čímž téměř odpadá kompilování PHP souborů, a

tak se výrazně zvyšuje výkon PHP aplikací. To však nic nemění na faktu, že bez eAcceleratoru je Zend relativně pomalý. Pokud by to tak nebylo, pak by to byl zřejmě ideální kandidát díky své výborně zpracované dokumentaci a díky tomu, že tento framework má podporu od vývojářů jádra PHP.

Ze srovnání [18] mi jako vítězové vyšly CodeIgniter a Nette. CodeIgniter má výbornou dokumentaci, z testovaných frameworků je nejrychlejší, stále však podporuje PHP4. Na oficiálním webu lze také vypozorovat, že jeho vývoj je poměrně pomalý. Navíc vývoj PHP4 byl již oficiálně ukončen, PHP5 se naopak již rozšířilo a je na téměř všech webových hostinzích. Tyto vlastnosti CodeIgniteru mě nakonec odradily. Nejedná se mi o žádné nesmyslné trvání na nejnovější verzi PHP. PHP5 má oproti PHP4 úplně přepsaný objektový model a obecně se až tato verze dá označit nálepkou „podporující objektové programování“. Dále přepsaný modul pro práci s MySQL a podporu pro výjimky. Zpracování chyb je tedy jednodušší. Jedná se tedy o zásadní změny, které velmi ovlivňují programování v PHP a do značné míry i výkon celé aplikace.

Zvolil jsem tedy Nette Framework, který má výbornou rychlost, výborné paměťové nároky a je založen na PHP5, přestože jeho dokumentace je dosti chudá. Tento nedostatek však částečně kompenzuje aktivní fórum, kde lze případné problémy poměrně rychle vyřešit. Velmi dobrým přehledem vlastností Nette je [20]. Vypíchl bych především ladící nástroj Laděnka, který poskytuje přehledný výpis volání na zásobníku (angl. stack trace) s hodnotami parametrů jednotlivých funkcí a dále i obsah systémových proměnných. Dále nativní podporu pro návrhový vzor Model-View-Presenter (MVP), který vede k oddělení aplikačního kódu a jeho grafické prezentace.

Kapitola 8

8 Editor pro vytváření kol

V této kapitole bych rád rozebral implementaci, možnosti, výhody a nevýhody webového editoru kol.

8.1 Implementace

Pro vytváření nových kol (často označované i jako levely) pro Sokobana i jeho alternativy jsem vytvořil editor kol (dále jen „Editor“) založený na PHP, JavaScriptu a na kaskádových stylech (CSS).

8.2 Koncept Editoru

Editor, který je možné si prohlédnout na obr. 6, se skládá z:

1) hrací desky,
2) textového pole pro název kola,
3) nastavení kreslícího módu,
4) výběru aktuálního „kreslícího“ objektu,
5) tlačítek pro uložení, či smazání daného kola,
6) textového pole pro komentář k danému kolu,
7) tlačítka pro smazání obsahu hrací desky a tlačítka pro zkontrolování herní desky,
8) tlačítek ovládajících velikost kola,
9) tlačítek pro posunutí objektů na hrací desce a
10) výběru herní varianty.

8.2.1 Vykreslování a mazání objektů z hrací desky

Nyní si popišme, jak se s Editorem pracuje. Základem je nanášení herních objektů na hrací desku. To provádíme tak, že si vybereme kliknutím libovolný objekt ze skupiny (4), následně klikneme na libovolné políčko na hrací desce a objekt je na hrací desku umístěn. Pokud si najedeme myší nad objekt ze skupiny (4), pak se nám po chvíli objeví nápověda, která obsahuje jméno objektu. Jeden z objektů, typicky umístěný vpravo, je označen jako „výchozí“ (v Editoru anglicky: Default – Erasing tool), jeho vybráním a postupným klikáním je možné odstranit z hrací desky jednotlivé objekty, které tam byly dříve vloženy. Rychlejší mazání je možné provést pomocí kliknutím na tlačítko „Clear desk“ ze skupiny tlačítek (7).

Obr. 6. Editor pro vytváření kol

8.2.2 Kreslící režimy

Editor podporuje dva kreslící režimy. První režim, tzv. naklikávací, byl popsán v předchozím odstavci. Tento režim je však nevhodný pro zaplnění velké plochy stejnými objekty – typicky jde o nakreslení zdí na okraj bludiště – jelikož by to vyžadovalo příliš mnoho kliknutí. Proto je možné zapnout kreslící režim (angl. Paint mode) a následně nanášet objekty na hrací desku pouhým pohybováním myši. Dále je možné kreslit tak, že budeme držet levé tlačítko myši a budou nanášeny objekty na hrací desku stejně jako v kreslícím režimu.

8.2.3 Nastavení velikosti bludiště

Nyní se podívejme, jak je možné nastavovat velikost bludiště a jaká jsou omezení. Upravovat rozměry bludiště je možné dvěma způsoby. Prvním způsobem je pomocí myši, kdy klikáním na tlačítka se šipkami ve skupině (8) vedle popisku „Velikost:“ (angl. size) můžeme každým kliknutím hrací desku zvětšit či zmenšit o jedno políčko v horizontálním či vertikálním směru. Horizontální šipky určují, jak se změní šířka bludiště (tj. zda se bludiště o jedno políčko prodlouží nebo zkrátí). Představme si, že levý horní roh bludiště je zafixován, pak je jasný význam šipky doleva, která bludiště zkrátí o jedno políčko na šířku. Ze stejné představy vychází i význam ostatních šipek, tedy například šipka nahoru zmenší výšku bludiště o jedno políčko.

Druhým způsobem je použití klávesových zkratek. Zkratky jsou napsány vždy v nápovědě pro dané tlačítko. Pro změnu velikosti bludiště se používá klávesová zkratka CTRL + <šipka>, stisknutí klávesové zkratky CTRL + <šipka doprava> má tedy stejný význam jako kliknutí na šipku doprava z ovládání šířky bludiště z minulého odstavce. Klávesové zkratky jsou zvlášť užitečné pro nastavení výšky bludiště, kdy by tlačítko s šipkou nahoru či dolů uskakovalo.

Ještě je třeba poznamenat, že pokud bludiště zmenšujeme, pak objekty, které jsou v nejpravějším sloupci bludiště (resp. objekty, které jsou umístěny na nejspodnější řádce bludiště) budou při zmenšování šířky (resp. výšky) trvale odstraněny z hrací desky. To znamená, že objekty, které se dostanou vlivem zmenšování hrací desky za hranice hrací desky, nejsou již nikde uschovány tak, aby se po opětovném zvětšení bludiště tyto objekty opět objevily na hrací desce. Platí jednoduché pravidlo, že existují pouze ty objekty, které vidíte na hrací desce.

8.2.4 Hromadné posunování objektů na hrací desce

Při vytváření bludiště se často stává, že je najednou potřeba posunout všechny objekty na hrací desce o několik políček jistým směrem.

Ovládání je podobné jako u nastavování velikosti bludiště. Je možné posouvat pomocí klikání na šipky vedle „Posunu“ (angl. Shift), směr šipky pak

jednoduše určuje, kterým směrem se všechny objekty na hrací desce posunou. Pro ovládání pomocí klávesnice se používá klávesová zkratka SHIFT + <šipka>.

Opět zde platí to, co platí i u nastavování velikosti hrací desky. Pokud se některé objekty posunou za hranice hrací desky, pak jsou nenávratně ztraceny a posunutí v opačném směru objekty na hrací desku nevrátí. Důvodem pro toto chování je to, že bychom mohli hrací objekty posunout o stovky políček za hranice hrací desky, ani bychom později o těchto objektech nevěděli, kde jsou, a pouze by se zpomalovala práce JavaScriptu, jelikož s políčky se pracuje jakožto s maticí.

Možnost posunování objektů na hrací desce by nebyla potřeba, pokud by rozšiřování a zkracování hrací desky fungovalo odlišně, a to tak, že by bylo možné natahovat a zkracovat hrací desku na libovolné straně bludiště. Obě řešení jsou však ekvivalentní a jde pouze o dva odlišné pohledy na stejnou věc. Dokonce i počet tlačítek by zůstal stejný, jelikož pro každý směr bychom museli mít dvě tlačítka – jedno pro zkracování, jedno pro rozšiřování.

8.2.5 Název kola a komentář

Název kola je vhodné volit uvážlivě, protože se bude zobrazovat ostatním hráčům v seznamu kol u dané ligy v hlavní aplikaci a v seznamu kol na webu, které daný hráč vytvořil. Samozřejmě je však možné kolo přejmenovat. Očekávané názvy kol jsou „Round01“ či „Level01“ apod. Komentář slouží spíše pro orientaci pro tvůrce kol. Je možné si například poznamenat, že je dané kolo rozpracované.

8.2.6 Výběr herní varianty

Ve výběru (10) je seznam všech herních variant, které zpřístupnil správce webu pro Editor. Změnou varianty se smaže obsah celé herní desky a načtou se herní objekty, dostupné pro danou variantu, do skupiny (4).

Různé varianty se mohou lišit jak pravidly pro počet objektů, tak i objekty, které je možné na hrací desku umísťovat, proto je nutné smazat obsah hrací desky při změně herní varianty.

8.2.7 Kontrola korektnosti bludiště

Tlačítko „Zkontrolovat korektnost“ (angl. „Check correctness“) ve výběru (7) slouží pro zkontrolování, že jsou splněny veškeré podmínky, které jsou dané danou herní variantou ve výběru (10).

Po kliknutí na tlačítko vyjede ze spodní části panelu s nastavením v Editoru informační panel s informacemi o korektnosti hrací desky. Typickými vlastnostmi, které se kontrolují, je počet objektů daného typu. Ať už v závislosti na jiných objektech či ne. Například objekt Sokoban se může vyskytnout pouze jednou na hrací desce v základní variantě (angl. ordinary).

Závislost na jiném objektu je možné demonstrovat na závislosti počtu beden na počtu cílových políček – oba počty musí být stejné v základní variantě.

8.2.8 Uložení kola

Uložení kola provádí ještě před samotným ukládáním kontrolu korektnosti kola. Pokud je kolo korektní, pak se teprve kolo skutečně uloží.

Pokud není kolo uloženo před odchodem z webové stránky Editoru, pak se objeví dotaz, zda skutečně se mají změny ignorovat a má se přejít na další webovou stránku. Toto upozornění chrání hlavně před nechtěným zavřením záložky ve webovém prohlížeči.

8.2.9 Smazání kola

Tlačítko (5) nám dává možnost kolo úplně smazat a přesunout se na seznam vytvořených kol. Tato volba je ekvivalentní s kliknutím na „Smazat“ (angl. Delete) v seznamu kol a je nevratná.

8.3 Výhody a nevýhody implementace na webu

Hlavním důvodem pro toto řešení byl fakt, že takto zůstane Editor dále snadno a rychle rozšiřitelný a také, že vytváření kol není vázané na nainstalování aplikace. Díky tomu, že JavaScript je napsán přenositelně, je možné, aby hráči vytvářeli kola i na jiných platformách než je Windows. Jmenovitě například na platformě Mac v prohlížeči Safari, či v Opeře na Linuxu. Přenositelnost JavaScriptu ne nutně znamená, že se používají pouze standardizované funkce a vlastnosti, příkladem je property innerHTML, která je implementována ve všech majoritních prohlížečích, avšak bude až součástí připravovaného standardu HTML5.

Toto řešení má i své nevýhody. Nevýhodou z pohledu hráče vytvářejícího nová kola může být fakt, že aby své kolo vyzkoušel, tak jej musí na webu uložit, otevřít aplikaci, dohledat dané kolo a spustit jej. Pokud by editor byl integrován do aplikace, šlo by stejného efektu dosáhnout tlačítkem „Otestovat“.

8.4 Detaily implementace

Jak již bylo řečeno, Editor jsem naprogramoval v PHP, JavaScriptu a vzhled jsem dopracoval pomocí kaskádových stylů (CSS).

8.4.1 JavaScript

JavaScript, což je interpretovaný jazyk, jehož skripty se provádí až na straně klienta (přesněji webového klienta, kterému posílá data webový server) ve webovém prohlížeči. Primárním účelem tohoto jazyka je dynamizace grafické podoby webové stránky, omezeně by bylo možné dosahovat stejného

efektu posíláním dalších požadavků webovému serveru, bylo by to však velmi neefektivní z pohledu přenosu dat na síti i z pohledu zbytečného zatěžování webových serverů.

Editor využívá pro svou funkčnost JavaScript na vše až na ukládání kol do databáze a načítání XML schémat.

8.4.2 Architektura Editoru

Na obr. 7 (na následující straně) je možné si prohlédnout schéma architektury Editoru. Schéma popisuje jednotlivé akce od otevření webové stránky uživatelem (označeno jako „Start“) až po opuštění webové stránky (označeno jakožto „Konec“). Schéma popisuje jednotlivé procesy, které po sobě následují. Obdélníky vyznačují proces, kosočtverce rozhodnutí, jednosměrná šipka popisuje přechod k dalšímu procesu či rozhodnutí. Obousměrná šipka značí požadavek směřovaný straně s vybarvenou šipkou a výsledek je vrácen straně s nevybarvenou šipkou.

Procesy označené oboustrannými šipkami se provádí dříve než procesy označené jednostrannými šipkami, což znamená, že proces „Načti kolo“ v procesu „PHP“ nejdříve požádá databázi o data a pak až připraví data pro Nette šablonu.

8.4.3 Použité JavaScriptové třídy a knihovny v Editoru

Editor se skládá z následujících komponent, které jsou uloženy ve formě JavaScriptových skriptů ve složce /www/js/.

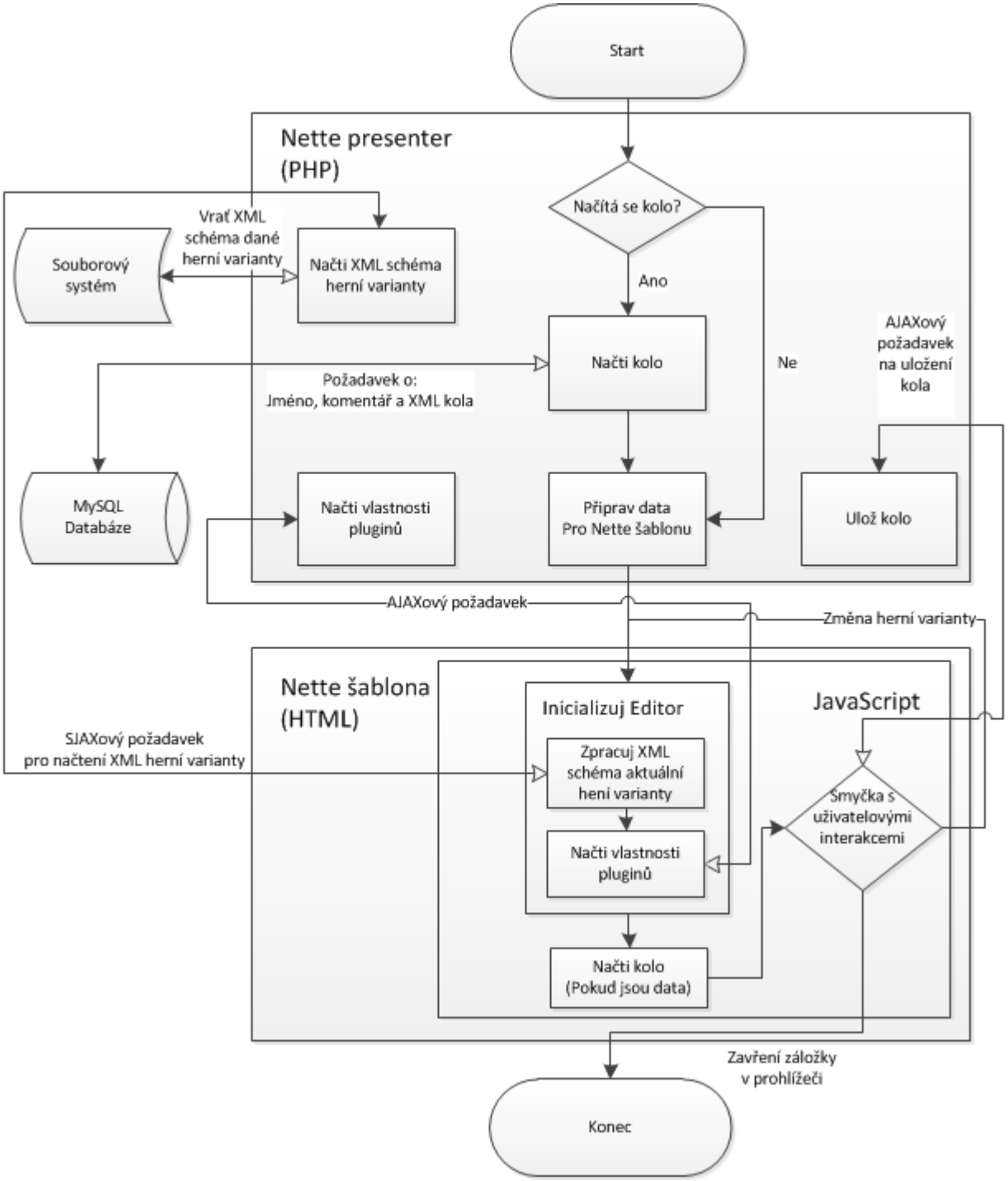
8.4.3.1 DomLib

DomLib je třída obsahující funkce pro práci s *Document Object Modelem* (DOM). Dle příslušné normy [21] můžeme DOM definovat jako: „Platformě a jazykové nezávislé rozhranní, které dovoluje programům a skriptům dynamicky přistupovat a upravovat obsah, strukturu a styl dokumentů. Dokument může být dále zpracován a výsledky zpracování mohou být začleněny zpět do současné stránky.“ Dokumentem se pak míní HTML, XHTML či XML.

Pomocí DOMu tedy můžeme upravovat kompletně vzhled a obsah HTML dokumentu, což je využíváno v Editoru například pro vkládání herních objektů do hrací desky, k rozšiřování hrací desky a mnoha dalším činnostem.

Tato knihovna obsahuje dvě funkce (dostupné v [22]) od Johna Resiga (jmenovitě *regEvent* a *removeEvent*), které řeší zaregistrování události u daného HTML elementu v JavaScriptu.

Problém, který tyto funkce řeší, spočívá v tom, že jednak názvy událostí ve webovém prohlížeči Internet Explorer (IE) začínají prefixem „on“ (příklady: onclick, onmousedown, …), kdežto v ostatních prohlížečích tento prefix chybí. Toto je triviální problém.



Obr. 7. Schéma architektury Editoru

Druhý problém je však závažnější, obecně je v JavaScriptu vždy kontextem (vyjádřený klíčovým slovem *this*) „vlastník“ dané funkce, tedy objekt, v němž je funkce definována, případně *Window*, pokud jde o globální funkci.

V Internet Exploreru však označuje klíčové slovo *this* v obslužné metodě, zavolané v rámci obsluhy události, objekt *Window*, přičemž v jiných prohlížečích je kontextem objekt, jehož událost obsluhujeme.

Jednoduchým důsledkem je fakt, že v Internet Exploreru nevíme, na kterém objektu byla událost vyvolána, pokud má více objektů stejnou obslužnou funkci. Řešení pana Resiga spočívá v tom, že funkce, která událost zpracovává, se přikopíruje k HTML objektu a pak již má klíčové slovo *this* správný kontext. Problém by nevznikal, pokud by se funkce *attachEvent* (v IE) chovala stejně jako standardizovaná funkce *addEventListener*.

Obě funkce jsem mohl naprogramovat sám, pokud bych si dříve přečetl [23], kde se velmi srozumitelně píše o problému a v podstatě i o řešení.

Další funkce obsažené v knihovně, které jsem již programoval já, slouží ke zjednodušení práce s DOMem. Jde o funkce *removeAllChildren*, *removeLastChild*, *getElementsByClassName*, *hasClass*, *addClass*, *removeClass*.

Dále knihovna obsahuje rozšíření funkcí Johna Resiga o jednodušší odregistrování událostí. V JavaScriptu totiž odregistrování události vyžaduje jako parametr funkci, která je zaregistrována. To je pochopitelně svazující a vyplatí se vytvořit si funkce, které si s tím poradí. Funkce pro zaregistrování události se jmenuje *regEventEx*, která funguje stejně jako funkce *regEvent* a navíc ukládá registrovanou funkci do pole (bufferu) *eventHolder*, který se přidá jako objekt k HTML objektu, jehož událost registrujeme. Samotný HTML objekt si tedy nese informaci o tom, jaké události jsou zaregistrované. Toho využívá funkce *removeEventsByTypeEx(el, type)*, která odstraní všechny události daného typu z daného HTML objektu s pomocí informací z bufferu *eventHolder*.

Veškeré funkce knihovny DomLib by zřejmě šly nahradit s pomocí knihovny jQuery, která se specializuje na to, že odbourává rozdíly mezi jednotlivými implementacemi JavaScriptu ve webových prohlížečích. Kód této knihovny je pro mě však tak komplikovaný, že jsem si raději požadované funkce naprogramoval, či někde našel (viz funkce *regEvent* a *removeEvent*), než abych měl knihovnu, ke které bych se musel chovat jako k černé skříňce.

8.4.3.2 Json2

JSON (JavaScript Object Notation) je odlehčený formát pro výměnu dat mezi různými službami, aplikacemi či platformami. Mezi výhody patří především to, že jde o textový formát, který je navíc velmi jednoduchý. Kompletní gramatiku, popis a stáhnutelné implementace pro různé programovací jazyky je možné najít v [24].

Pro představu přidávám jednoduchý příklad možného popisu knihy:

```
{
    "autor" : {
      "jmeno" : "Jaroslav",
      "prijmeni" : "Reichl"
    }
    "nazevKnihy" : "Klíč k fyzice",
    "nakladatelstvi" : "Albatros",
    "rokVydani" : 2005,
    "ISBN" : "80-001590-0",
    "pocetStran" : 220
 }
```

Json2 je skript, který je veřejně dostupný na webové adrese [25] a který emuluje podporu pro JSON v JavaScriptu, pokud jej daný prohlížeč nepodporuje nativně. Moderní majoritní webové prohlížeče však JSON nativně podporují. Tento skript je tedy přidán pouze pro zajištění kompatibility s ostatními prohlížeči, či staršími verzemi majoritních webových prohlížečů.

8.4.3.3 RoundEditor

Tato knihovna tvoří jádro Editoru. Základ tvoří metody, které upravují vybranou tabulku v HTML, tedy přidávají do herní objekty, rozšiřují ji, zkracují ji a pohybují herními objekty na ní. Tedy *RoundEditor* vyžaduje pro své fungování HTML element *table* – jde také o parametr konstruktoru třídy *RoundEditor*.

8.4.3.3.1 Kreslení

Buňky dané tabulky, tedy *td* elementy, mají přiřazen atribut *id* s hodnotami *cell-<radek>x<sloupec>*. Díky tomu je možné snadno adresovat každou buňku v tabulce pomocí standardní funkce *getElementById* v JavaScriptu. Tím jsme tedy získali matici.

„Kreslení“ do tabulky je prováděno pomocí upravování buněk tabulky. Jelikož herní objekty, kterými můžeme kreslit, jsou rozděleny na „podloží“ (angl. tile) a objekty, tak existují i dva způsoby kreslení. Pro vykreslení podlaží jsem použil CSS vlastnost buňky *background-image*, která vykreslí obrázek jako pozadí buňky. Objekty se vykreslují pomocí přidání HTML elementu *img* (HTML značka pro obrázek) do buňky. V obou případech se používají základní funkce DOMu. V *RoundEditoru* se používá funkce *PaintOnField* pro přidání „podloží“ nebo objektu do interních struktur *RoundEditoru* a až funkce *RedrawField* zajistí skutečné zobrazení objektu či „podloží“ na hrací desce.

Všechny obrázky pro Editor mají 36x36 pixelů a buňky mají stejnou velikost, proto „nakreslení“ herního objektu do tabulky vypadá jako vykreslení herního objektu a ne jako přidání obrázku do buňky jakožto kontejneru.

Rozšíření herní desky se děje přidáním elementu *td* do každého řádku tabulky, případně přidáním celé nové řádky, tj. elementu *tr* a příslušného počtu elementů *td* do řádky. *RoundEditor* pro tyto události obsahuje funkce *appendCell*, respektive *appendRow*.

8.4.3.3.2 Načítání herní varianty

Herní varianty jsou uloženy ve složce /www/js/RoundEditor/Schemas/. Každá varianta je složená z XML dokumentu, který ji plně popisuje a z obrázků umístěných ve složkách *<Název varianty>Resources*.

V *RoundEditoru* se načítání herní varianty provádí pomocí funkce *LoadXmlScheme*. Na obr. 7 pak proces „Zpracuj XML schéma aktuální herní varianty“ odpovídá této funkci.

Pro vysvětlení herního schématu zde předkládám zápis standardní varianty Sokobana.

```
<?xml version="1.0" ?>
<Scheme>
  <DefaultTile>
    <Name>Default</Name>
    <Img>OrdinaryResources/obj_A_dark.png</Img>
  </DefaultTile>

  <Object>
    <Name>Sokoban</Name>
    <Img>OrdinaryResources/obj_S.png</Img>
    <MinCount>1</MinCount>
    <MaxCount>1</MaxCount>
    <Version>1.00</Version>
  </Object>

  <Object>
    <Name>Wall</Name>
    <Img>OrdinaryResources/obj_W.png</Img>
    <MinCount>0</MinCount>
    <MaxCount>Infinity</MaxCount>
    <Version>1.00</Version>
  </Object>

  <Object>
    <Name>Box</Name>
    <Img>OrdinaryResources/obj_B.png</Img>
    <MinCount>1</MinCount>
    <MaxCount>Infinity</MaxCount>
    <Version>1.00</Version>
  </Object>

  <Object>
    <Name>Aim</Name>
    <Img>OrdinaryResources/obj_A.png</Img>
    <MinCount>1</MinCount>
    <MaxCount>Infinity</MaxCount>
    <Conditions>
      <CondCountEqualAs>Box</CondCountEqualAs>
    </Conditions>
```

```
    <IsTile>Yes</IsTile>
    <Version>1.00</Version>
  </Object>

  <Object>  <!-- also erasing tool -->
    <Name>Default</Name>
    <IsTile>Yes</IsTile>
  </Object>
</Scheme>
```

První značkou ve schématu je *DefaultTile*, která určuje, jakými „podložími“ bude deska vyplněna před nakreslením čehokoli uživatelem. *DefaultTile* definuje jméno, které je stejné jako jméno u poslední značky *Object*, tím označíme, že poslední objekt je zároveň objekt, kterým se maže. Přeházením pořadím značek *Object* by se změnilo i vykreslení v Editoru v nabídce „kreslících objektů“.

Jak je možné si všimnout, značky *Object* definují svůj obrázek tagem *Img*, respektive cestu k němu, ta je relativní k cestě /www/js/RoundEditor/Schemas/.

Objekty jsou dourčeny značkami:

- *Name* – Jméno herního objektu. Dle tohoto údaje se řídí, kde bude hledáno XML schéma herního objektu, které potřebuje PHP backend Editoru. Cesta ke schématu je definována takto: /www/Download/Official/Plugin<Name>/v<Version>/XmlSchema.xml
- *Version* – Primárně pro určení cesty výše.
- *MinCount* – Určuje minimální počet výskytů herního objektu na hrací desce při kontrolování korektnosti kola či ukládání kola.
- *MaxCount* – Analogicky maximální počet. Speciální hodnotou je „Infinity“, která říká, že počet není omezen (je však omezen počtem políček na hrací desce).
- *IsTile* – Určuje způsob vykreslování herního objektu.
- *Conditions* – Způsob pro zpřísnění pravidel pro korektnost kola. Jediným implementovaným potomkem je *CondCountEqualAs*.
- *CondCountEqualAs* – Určuje, že herní objekt definující tuto podmínku se bude muset vyskytovat na hrací desce v takovém počtu jako obsah značky *CondCountEqualAs*.

Máme tedy vysvětlené schéma a nyní je ještě třeba rozebrat jeho reprezentace v *RoundEditoru*. Objekty ze schémata se uloží do JavaScriptových objektů *JsSchemaObject*, které mají stejné vlastnosti, jaké byly vyjmenovány výše.

GameObject je třída, jejíž instance představují jednotlivé herní objekty na hrací desce. V každé instanci je uložen index do pole *jsSchemaObjects* v instanci třídy *RoundEditor* a pole s vlastnostmi daného objektu. Mezi

vlastnostmi jsou *PosX, PosY, Version*, další vlastnosti jsou závislé na XML schématu, které si specifikuje každý plugin sám.

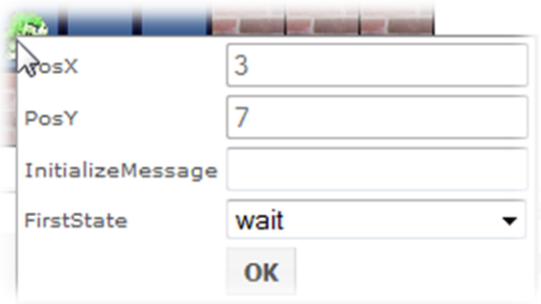
8.4.3.4 RoundEditorInstanceHandling

Jde o soubor funkcí, které většinou komunikují s instancí třídy *RoundEditor* a informují uživatele o výsledku.

Tyto funkce by mohly být umístěny i v HTML šabloně Editoru, ovšem tam by nebylo možné ladění, proto jsou tyto funkce v novém souboru.

8.4.3.5 PopUpForm

PopUpForm je třída pro vytváření tzv. vyskakovacích oken s formulářovými prvky (viz obr. 8), kterou jsem vytvořil pro nastavování vlastností jednotlivých herních objektů. Tento způsob nastavování jsem zvolil proto, že umožňuje nastavovat najednou vlastnosti více objektů zároveň a také šetří místo v Editoru, tím že využívá třetí dimenzi.



Obr. 8. Nastavování vlastností herních objektů v Editoru

8.4.3.6 MouseWrapper

MouseWrapper je pouze jednoduchá třída, která sjednocuje obsluhu událostí spojených s myší napříč webovými prohlížeči. Vytvářena byla třída primárně kvůli odchytávání kliků pravého tlačítka myši.

8.4.3.7 KeyboardWrapper

Tuto třídu v JavaScriptu jsem vytvořil pro zachytávání klávesových zkratek, které jsou použity v Editoru. Nejzajímavější metodou třídy je *RegisterShortcut*, která klávesové zkratce přiřadí obsluhu. Příklad použití:

```
keyboardWrapper.RegisterShortcut("ctrl+left", callback);
```

Je možné zachytávat klávesové zkratky ve tvaru:

1) modifikátor + „ne-modifikátor“
2) „ne-modifikátor“

Modifikátorem jsou klávesy CTRL, SHIFT, ALT (lze změnit v poli *modifyingKeys*), „ne-modifikátor“ jsou zbývající klávesy.

8.4.3.8 TransformEffect, DisplayEffect, FadeEffect

TransformEffect je třída, kterou jsem vytvářel, jelikož jsem potřeboval provádět lineární transformace vlastností DOM objektů. Tato třída staví své chování na standardní JavaScriptové funkci *setTimeout*, kterou lze použít pro opakované volání funkce po uplynutí určité časové periody.

FadeEffect je třída využívající *TransformEffect* pro změnu průhlednosti DOM objektu. Prohlédnout si její chování je možné při změně výběru „kreslícího“ objektu v Editoru.

DisplayEffect je rovněž třída založená na třídě *TransformEffect*. Je určená pro animování zobrazení a skrytí DOM objektu. Původně jsem chtěl využít pro tuto funkcionalitu knihovnu jQuery, problém však byl, že z nějakého důvodu ve webovém prohlížeči Firefox při animování docházelo k zobrazování a skrývání posuvníku (angl. scrollbar), což na pohled působilo velmi nevzhledně.

8.4.3.9 AjaxCall

AJAX (Asynchronous JavaScript and XML) je rozhranní v JavaScriptu, jakým můžeme získávat data od webového serveru, přičemž nedojde k obnovení obrazovky ve webovém prohlížeči. Na této technice je založeno mnoho moderních webů, které se tak přibližují vzhledem desktopovým aplikacím.

AjaxCall je jednoduchá třída, kterou jsem si vytvořil pro zaobalení vytváření AJAXových a SJAXových požadavků na webový server v JavaScriptu. První požadavek probíhá asynchronně (po zavolání se nečeká, až se požadavek dokončí a pokračuje se ve zpracovávání JavaScriptu; později až dojde k zavolání obslužné funkce a data se zpracují) a druhý synchronně.

Třída se skládá z:

- konstruktoru *AjaxCall(url, callback, isAsynchronous)*
  - callback je funkce se signaturou: function(responseText, responseStatus, responseXML)
- metody *Execute(passData, method)* – pro započetí (a)synchronního volání.
  - *passData* je ve formátu var1=val1&var2=val2, případně je možné poslat jako parametr i objekt a metoda si ho sama převede do tohoto tvaru
  - *method* – POST nebo GET
- metody *Cancel()* – pro zrušení probíhajícího asynchronního volání

8.4.4 Použité PHP knihovny pro Editor

V Editoru je použita jedna větší PHP knihovna, kterou jsem pojmenoval *SchemaNodes*. Knihovna umí vyčíst z XML schématu, jaké potomky vybraný element očekává, v jakém počtu a s jakými vlastnosti.

Pro pochopení dalšího textu je nezbytné si přečíst odstavec 5.4 o využití XML v herních objektech.

K napsání knihovny mě vedlo to, že každý herní objekt má své vlastní schéma, které určuje, jaká inicializační data potřebuje, bylo by tedy zbytečné do Editoru zadávat tyto údaje znovu, když jsou již takto dostupné.

V odstavci 5.4 se rozebíralo vytvoření XML schémata pro element *Monster*. Na této ukázce si můžeme demonstrovat, co knihovna *SchemaNodes* dokáže. Budeme chtít z této ukázky vyčíst potomky (inicializační data) elementu *Monster*. Zdrojový kód níže ukazuje použití třídy *SchemaNodes*.

```
<?php
  $schemaNodes = new SchemaNodes($XmlSchema,
     '/xs:schema/xs:element[@name="Monster"]');
  $schemaNodes->Load();
?>
```

Proměnná *$XmlSchema* obsahuje XML schéma, které vzniklo kombinací ukázky č. 2 a ukázky č. 3 z odstavce 5.4. Druhý parametr konstruktoru třídy je *XPath* pro vybraný element (v našem případě *Monster*). *XPath* je jazyk, pro adresování částí XML dokumentů, informace o tomto je jazyce je možné najít v oficiální dokumentaci [26].

Knihovna obsahuje ještě metodu *debugDump* pro výpis, jakým způsobem zpracovávala XML schéma. Její výstup je v tomto příkladu takovýto:

```
1 :: +parseElement [name=Sokoban; type=SokobanType]
 2 :: +parseBuiltInType [name=Sokoban; type=SokobanType]
  3 :: +parseComplexType [name=SokobanType]
   4 :: +parseComplexContent
    5 :: +parseExtension
     6 :: +parseComplexType [name=GameObjectType]
      7 :: +parseSequence
       8 :: +parseElement [name=Version; type=xs:string]
        9 :: +parseBuiltInType [name=Version; type=xs:string]
        10 :: -parseBuiltInType [name=Version; type=xs:string]
       11 :: -parseElement [name=Version; type=xs:string]
       12 :: +parseElement [name=PosX; type=xs:integer]
        13 :: +parseBuiltInType [name=PosX; type=xs:integer]
        14 :: -parseBuiltInType [name=PosX; type=xs:integer]
       15 :: -parseElement [name=PosX; type=xs:integer]
```

```
 16 :: +parseElement [name=PosY; type=xs:integer]
  17 :: +parseBuiltInType [name=PosY; type=xs:integer]
  18 :: -parseBuiltInType [name=PosY; type=xs:integer]
 19 :: -parseElement [name=PosY; type=xs:integer]
 20 :: +parseElement [name=InitializeMessage; type=xs:string]
  21 :: +parseBuiltInType [name=InitializeMessage; type=xs:string]
  22 :: -parseBuiltInType [name=InitializeMessage; type=xs:string]
 23 :: -parseElement [name=InitializeMessage; type=xs:string]
 24 :: -parseSequence
25 :: -parseComplexType [name=GameObjectType]
26 :: +parseSequence
 27 :: +parseElement
  28 :: +parseElement [name=FirstState]
   29 :: +parseSimpleType
    30 :: +parseRestriction
     31 :: +parseBuiltInType
     32 :: -parseBuiltInType
    33 :: -parseRestriction
   34 :: -parseSimpleType
  35 :: -parseElement [name=FirstState]
 36 :: -parseElement
 37 :: -parseSequence
 38 :: -parseExtension
 39 :: -parseComplexContent
 40 :: -parseComplexType [name=SokobanType]
41 :: -parseElement [name=Sokoban; type=SokobanType]
```

Tato knihovna samozřejmě nedokáže zpracovat libovolně složité XML schéma verze 1.0, zvládne pouze podmnožinu, která je nejčastěji používána. Pokud se použije nepodporovaná vlastnost v XML schématu, pak se jednoduše někteří potomci ve výsledku neobjeví.

Výsledek zpracování metody *Load* je uložen do pole *$schemaNodes->children*. V našem příkladu toto pole obsahuje potomky *Version*, *PosX*, *PosY*, *InitializeMessage* a *FirstState*. Na tomto je vidět, že výsledek obsahuje jednak potomky z typu *MonsterType*, ale i z typu *GameObjectType*, takže knihovna *SchemaNodes* evidentně podporuje i rozšíření typů.

V Editoru se výsledky této třídy používají jako vstup pro knihovnu *PopUpForm*. Zapojení třídy do Editoru je znázorněno na obr. 7 jakožto „Načti vlastnosti pluginů“.

Hotový testovací skript knihovny *SchemaNodes* je možné najít ve složce *tests* ve zdrojových kódech webové aplikace.

Kapitola 9

9 Statistiky

Tato kapitola popisuje, jakým způsobem je řešeno bodové ohodnocení hráčů za výhry.

9.1 Hra pro jednoho hráče

Ligy (angl. leagues; označované také jako „Quests“) jsou rozděleny na oficiální a neoficiální. Důvodem je to, aby si hráči nevytvářeli kola, která pak budou vyhrávat a budou za ně získávat body.

Pokud hráč vyhraje kolo z oficiální ligy, pak dostane za výhru 3 body. Pokud se mu navíc podaří překonat nejlepší čas, pak získává bodů 10.

Za vyhrané kolo z neoficiální ligy nedostane hráč body žádné.

9.2 Hra dvou hráčů

Jelikož je možné hrát dané kolo se soupeřem libovolněkrát, tak za hru dvou hráčů žádné body přirazeny nejsou.

9.3 Žebříček

Žebříček hráčů je vytvářen podle počtu bodů. Hráč se může porovnat se všemi ostatními hráči, tento žebříček se nazývá „International rankings“ nebo s hráči ze své země, na který je možné se podívat v sekci „Profile“.

Důvěryhodnost žebříčku není možné přesně stanovit, nicméně je možné, že jeden hráč se zaregistruje vícekrát a bude si kola řešit na jednom účtu a později na druhém, kde bude získávat snadno body. Tomuto se dá velmi obtížně zabránit, jelikož zkontrolovat, že dva účty ovládá skutečně pouze jeden hráč, není s IPv4 protokolem vůbec jednoduché – hlavním problémem jsou privátní IP adresy, pod kterými může hrát více různých hráčů. Důsledná kontrola hráčů a jejich pravých identit by byla kontraproduktivní, jelikož by odradila mnoho hráčů. Z těchto důvodů je koncepce statistik velmi jednoduchá.

Kapitola 10

10 Instalace

Tato kapitola popisuje způsob instalace webové aplikace a desktopové aplikace.

10.1 Desktopová aplikace

Hlavní aplikaci je možné nainstalovat pomocí instalátoru, který je dostupný na přiloženém médiu ve složce *Application/Install/*. Instalace programu probíhá standardním způsobem jako u většiny aplikací pro MS Windows. Stačí spustit soubor *setup.exe* a instalátor vás provede zbytkem celé instalace.

Pro hraní hry je nutné si nastavit ještě nastavit chování klávesnice, je nutné nastavit nízkou dobu prodlevy před opakováním stisku klávesy a rychlé opakování stisků kláves. Nastavení je možné nalézt ve Windows v *Ovládacích panelech* v sekci *Klávesnice*. Aplikace toto nečiní sama proto, že by tím změnila chování ostatních programů.

10.2 Webová aplikace

Webová aplikace se instaluje ve čtyřech krocích. Prvním je vytvoření databáze a databázových tabulek, jejichž struktura je uložena ve formě SQL ve složce *Web/db-dump.sql*.

Druhým krokem je nakopírování zbývajících souborů ve složce Web na webový server, typicky prostřednictvím FTP protokolu. Předposledním krokem je editace nastavení souboru *Web/app/config.ini*, který obsahuje údaje o připojení k databázi v sekci [production < common].

Posledním krokem je přiřazení práv k zápisu složkám *app/temp* a *app/log*, tak aby PHP skripty mohly do těchto složek zapisovat.

V tomto odstavci se předpokládá, že čtenář bude schopen instalaci webové aplikace zvládnout. Vzhledem k tomu, že by to měli provádět lidé, u kterých se předpokládají základní znalosti PHP a MySQL.

Kapitola 11

11 Uživatelská příručka

Tato kapitola popisuje ovládání programu a ovládání webové aplikace pro běžného uživatele.

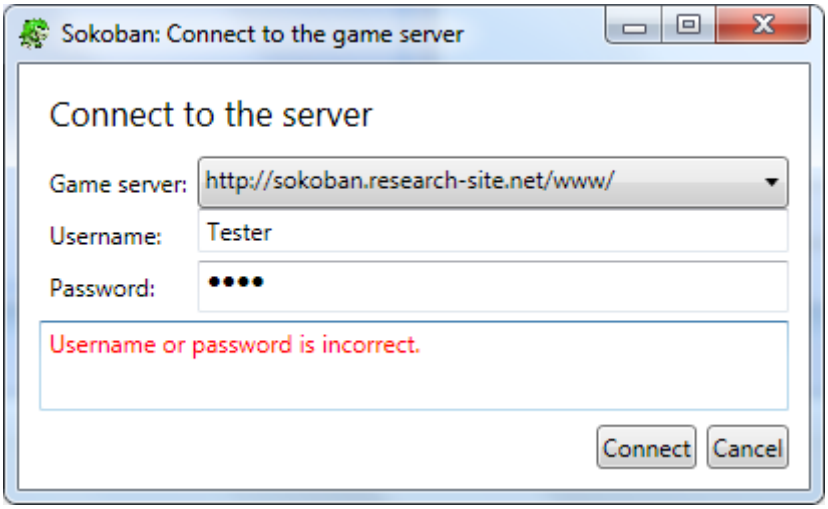
11.1 Spuštění programu

Program lze spustit ze složky, kam byl nainstalován (typické umístění je *C:\Program Files\SoTh*) dvojitým kliknutím na soubor *Sokoban.exe*. Další možností je otevření kliknutím na ikonu programu na ploše. Případně kliknutím na tlačítko *Start*, kliknutím na *Všechny programy* a otevřením složky *SoTh* a kliknutím na *SoTh*.

11.2 Přihlášení do aplikace

Po spuštění programu se otevře dialogové okno (viz obr. 9), které požaduje přihlašovací údaje, tedy přihlašovací jméno a heslo, a herní server, na kterém chcete hrát. Pro přihlášení je tedy nutné se na herním serveru zaregistrovat.

Registraci na herním serveru [27], který je součástí této práce, lze nalézt v nabídce pod logem pod označením „Sign Up“ (zaregistrovat se). Registrace spočívá pouze ve vybrání uživatelského jména, hesla a vyplnění e-mailové adresy a země, za kterou chcete hrát. Po úspěšné registraci je možné zadat přihlašovací údaje do dialogu v aplikaci a pod touto identitou budete v aplikaci přihlášeni.



Obr. 9. Přihlašovací dialog aplikace

Při přihlašován dochází ke kontaktování vzdáleného serveru, proto rychlost přihlášení závisí na konkrétním připojení k Internetu a rychlosti herního serveru. Může se tedy stát, že aplikace bude zdánlivě nečinná po určitou dobu.

Pokud se přihlášení nezdaří, pak je popsán důvod, jak je možné si prohlédnout na obr. 9.

Bez přihlášení není možné využívat žádné funkce programu, proto při snaze o vypnutí přihlašovacího dialogu budete vyzváni, zda skutečně chcete aplikaci vypnout.

Pro účely testování byl vytvořen testovací účet s přihlašovacím jménem *Tester* a heslem *philips*. Pod tímto účtem je tedy možné aplikaci vyzkoušet v případě, že se nechcete registrovat.

Dialog pro přihlašování se zobrazuje při každém spuštění aplikace. Přihlašování je možné urychlit spuštěním aplikace s parametry:

Sokoban.exe --auto-connect <herní server> <uživatelské jméno> <heslo>

11.3 Spuštění hry pro jednoho hráče

Hru jednoho hráče je možné hrát ve dvou režimech. Buď hraním jednoho kola, či hraní celé ligy (označované také jako *quest*).

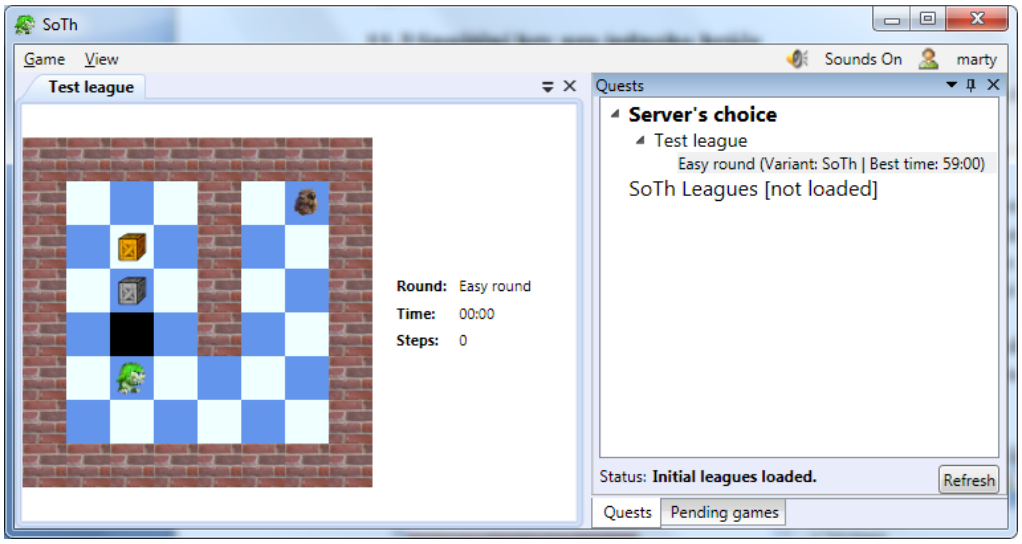
V panelu *Quests* jsou k dispozici kategorie, ligy a kola, které je možné hrát. Jedna kategorie je tvořena množinou lig a liga je tvořena množinou kol.

Tučným písmem jsou označeny kategorie, které jsou načteny, tedy kategorie, jejichž ligy jsou načtena z herního serveru do aplikace. Kategorie, které nejsou načteny, mají v názvu postfix *[not loaded]*. Tyto kategorie je možné načíst dvojitým kliknutím na ligu. Podobně to je pro načtené a nenačtené ligy. Na obr. 10 si lze vidět načtenou kategorii *Server's choice* a nenačtenou kategorii *SoTh Leagues*.

Důvodem pro toto ovládání je snaha o šetrné stahování dat z herního serveru. Herní server má nastaveno, které kategorie a ligy budou rovnou načteny do aplikace a které ne.

Hra pro jednoho hráče se spouští dvojitým kliknutím na jméno ligy, pro hraní všech kol ligy, či dvojitým kliknutím na jméno kola, pro hraní daného kola. Je možné spustit ligu či kolo kliknutím pravého tlačítka na jméno ligy či kola a vybráním *Play league*, resp. *Play round*.

Hra se spustí v okně pro hraní pod novou záložkou. Je možné si najednou otevřít více záložek s hrami, avšak spuštěná je vždy pouze aktuálně otevřená hra.



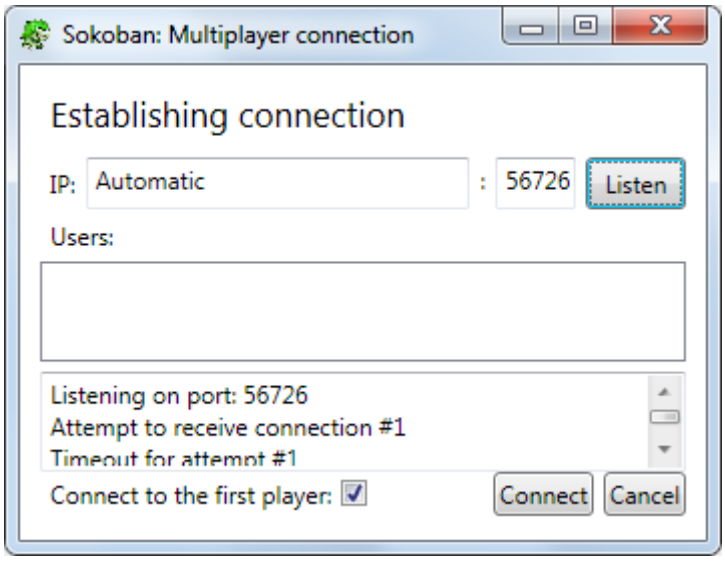
Obr. 10. Načítání kategorií a lig.

11.4 Spuštění síťové hry

Síťová hra začíná nabídkou hry prvním hráčem, přijmutím hry druhým hráčem a spuštěním hry samotné.

11.4.1 Výzva k síťové hře

Výzvu ke hře je možné zadat kliknutím pravého tlačítka myši na libovolné kolo v panelu *Quests* a vybráním možnosti *Play over network* z kontextové nabídky. Následně se zobrazí dialogové okno zobrazené na obr. 11.



Obr. 11. Dialog pro navázání spojení

Nyní si popišme jednotlivé části dialogového okna podrobněji.

- **IP a port** – IP adresa a port, na kterých bude síťové spojení probíhat. Jde o protokol IPv4. Pokud IP adresa obsahuje speciální

hodnotu „Automatic“, pak bude očekáváno příchozí spojení na všech síťových rozhranních najednou, pro méně zkušené uživatele doporučuji tyto hodnoty neměnit.

- **Users** – Do tohoto pole se přidávají hráči, se kterými se podařilo navázat síťové spojení, pokud však není zaškrtnuta volba „Connect to the first player“ (Připojit se k prvnímu připojenému hráči).
- **Listen** – Toto tlačítko spustí navazování síťového spojení na dané IP adrese a portu. Zároveň se kontaktuje herní server a vloží se na něj nabídka ke hře. Průběh navazování spojení je popisován ve stavovém okně, jak je možné vidět na obr. 11. Navazování spojení je možné zrušit kliknutím na tlačítko „Cancel“, na které se změní tlačítko „Listen“ po jeho stisknutí.
- **Connect to the first player** – Tato volba značí, že hra bude navázána s prvním hráčem, se kterým se podaří navázat spojení.
- **Connect** – Pokud není zaškrtnuta volba „Connect to the first player“, pak stisknutí tohoto tlačítka vede k spuštění hry s vybraným hráčem v nabídce „Users“.

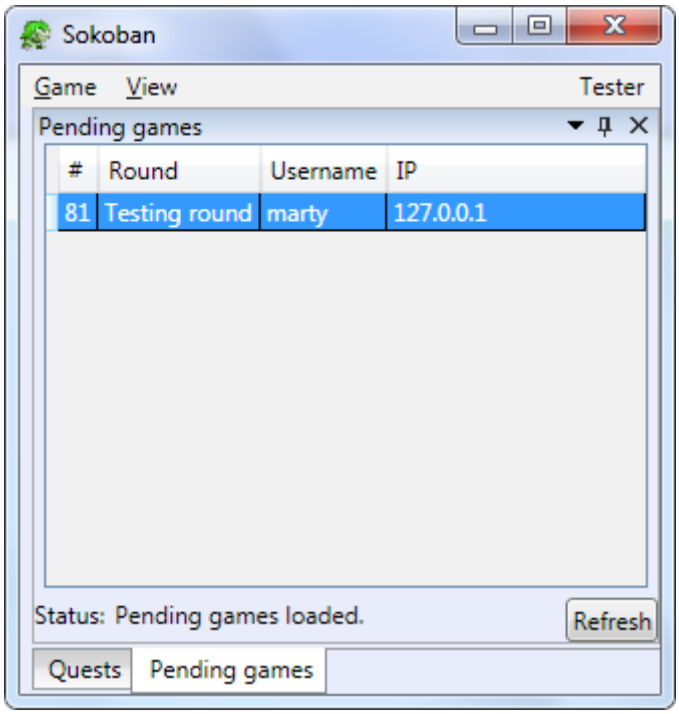
Volba *Connect to the first player* není pouze drobnost, která ušetří čas hráči. Tato volba je důležitá, protože rozlišuje, zda se bude hráč v roli vyzývatele snažit o navázání více spojení najednou či ne, spojení s jedním uživatelem je vždy stabilnější.

Problém při připojování nastává v případě, že vyzývatelem je hráč, který má privátní IP adresu a nemá nastavené přesměrování portu (angl. port forwarding) pomocí překladu síťových adres (angl. Network Address Translation), pak připojení nebude navázáno.

Zda má uživatel privátní IP adresu, se může uživatel přesvědčit srovnáním IP adresy ve Windows v dialogovém okně s připojením, kde je uvedena IPv4 adresu připojení, a IP adresy, kterou hlásí webová služba [28]. Shodují-li se, pak uživatel má veřejnou IP adresu, neshodují-li se, tak privátní. Pro privátní IP adresy je třeba nastavit překlad síťových adres. Jak se toto dělá, je možné dohledat v manuálu k vašemu routeru.

11.4.2 Přijmutí výzvy

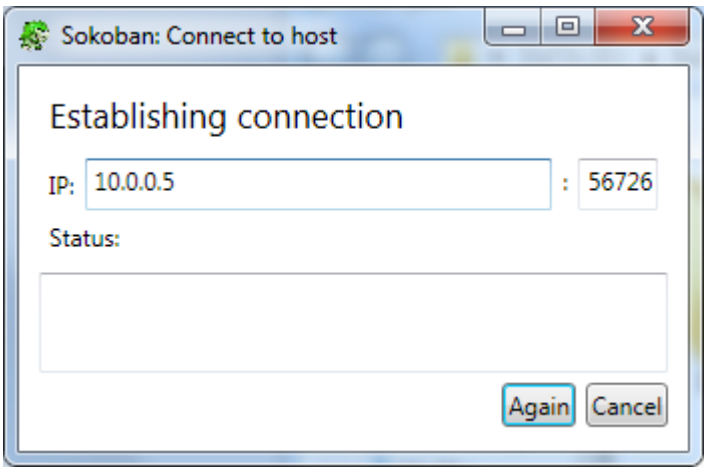
Výzvu je možné přijmout v hlavní aplikaci v panelu *Pending games* (nabídky ke hře; viz obr. 12) dvojitým kliknutím na nabídku. V tomto panelu se zobrazují nabídky, které byly vloženy na herní server vyzývateli.



Obr. 12. Přijímání nabídky ke hře

Následuje zobrazení dialogu pro připojení k vyzyvateli, který je možné si prohlédnout na obr. 13. Dialog již bude mít předvyplněnou IP adresu a port z nabídky zadané vyzývatelem. Po otevření dialogu se kliknutím na tlačítko „Connect“ je možné pokusit se připojit k vyzývateli hry. O průběhu této snahy je hráč informován v poli „Status“.

Po navázání spojení začne hra, jak bylo uvedeno v odstavci 6.1. Hru je možné ukončit jednoduchým vypnutím záložky s hrou, případně dohráním hry, pak je síťové připojení ukončeno.



Obr. 13. Dialog pro připojení k vyzyvateli

11.5 Používání řešitelů

Používání řešitelů v aplikaci je velmi jednoduché. Řešené kolo je vždy to, které zrovna vidíte v programu. Jelikož panel s jednotlivými koly může být zavřený, tak přesněji by se mělo říci, že řešené je to kolo, které bylo naposledy označené jako aktivní. Na obr. 14 je aktivní okno DebugQuest.

11.5.1 Spuštění řešitele

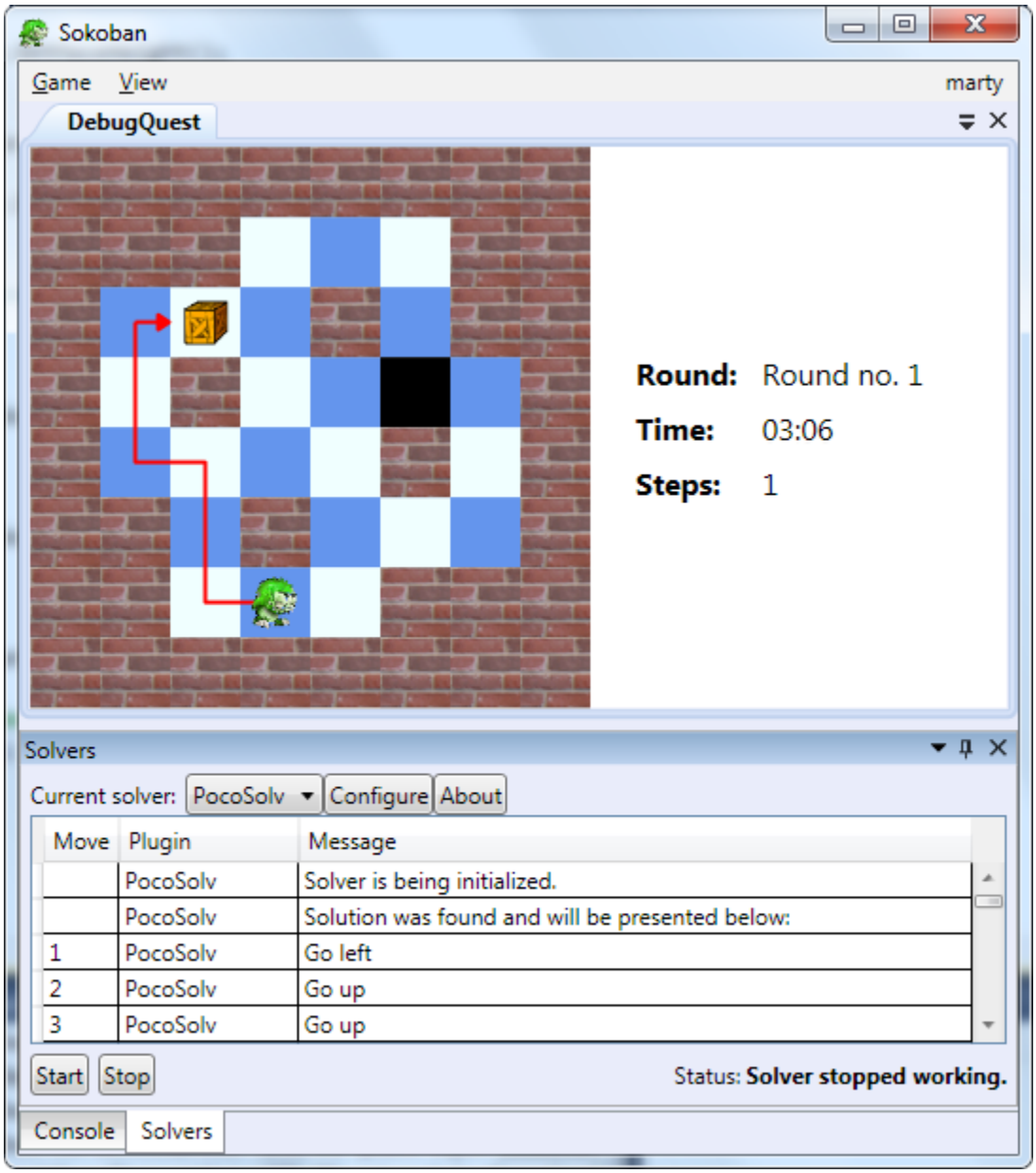
1. Vyberte řešitele, kterého chcete pro řešení použít. Na obr. 14 je vybraným řešitelem *PocoSolv*.
2. Klikněte na tlačítko „Start“. Informace o průběhu práce řešitele je možné sledovat vpravo dole u popisku „Status“.

Poznámka: Po zapnutí řešitele je možné přepínat mezi koly a nechat řešitele pokračovat v práci.

Je možné hledat i řešení od uživatelů pomocí tlačítka „Get solution from game server“ („Najdi řešení na herním serveru“). Jde o obdobu tlačítka „Start“, jen tato akce hledá řešení mezi řešeními, které nashromáždili hráči, kteří kolo dohráli. Vrátí se jedno náhodné řešení. Hledání řešení nelze vypnout.

11.5.2 Vypnutí řešitele

Jestliže řešitel neskončil svou práci (tlačítko „Start“ bude neaktivní a status bude „Solver is running“) a vy požadujete vypnutí řešitele, pak klikněte na tlačítko „Stop“. Řešitel se pokusí ukončit svou činnost, jakmile k tomu bude mít možnost, ovšem za předpokladu, že tuto funkci podporuje. Ať již tuto funkci podporuje nebo ne, objeví se o tom informace v tabulce se zprávami od řešitele v panelu *Solvers*.



Obr. 14. Práce s řešiteli v hlavním programu

11.5.3 Zobrazení řešení

Po úspěšném vyřešení kola se zobrazí v tabulce se zprávami od řešitele na jednotlivých řádcích řešení v textové podobě a v bludišti v řešeném kole se pak objeví šipka, která představuje grafické znázornění části řešení, která vede k prvnímu posunu bedny a naznačuje, jakým směrem má být bedna posunuta.

Podívejme se nyní na obr. 14. Pokud se nyní Sokoban posune vpravo (obecně jiným směrem než naznačuje červená šipka), tak vyznačené řešení zmizí a řešitel není znovu automaticky zapnut. Takové chování je, dle mého názoru, pro hráče logické.

Pokud se v situaci na obr. 14 Sokoban posune vlevo, pak část vyznačeného řešení, po které Sokoban již prošel, zmizí, aby řešení na hrací desce bylo i

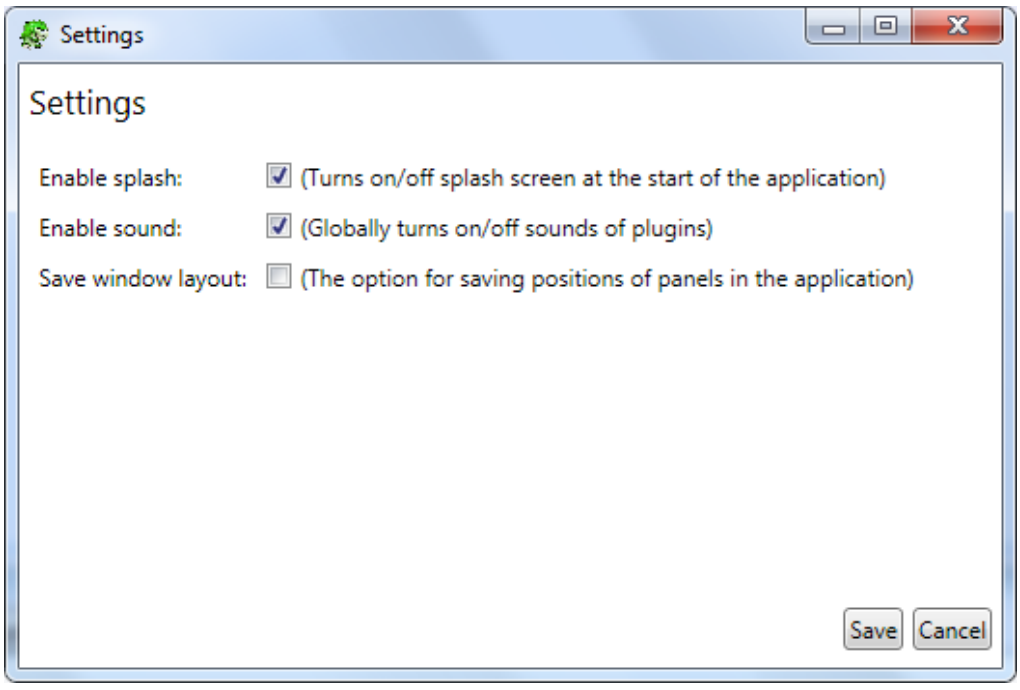
nadále přehledné a také proto, aby zůstala jasná pravidla pro to, kdy řešení zmizí, jak již bylo probíráno v minulém odstavci.

Konečně pokud se Sokoban přesune na políčko se souřadnicemi [3,3] (tj. posune bednu), tak se objeví nová šipka, která bude prezentovat další pohyby Sokobana až po další první posouvání bedny.

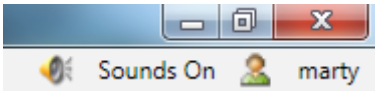
11.6 Nastavení aplikace

Nastavení aplikace obsahuje pouze pár údajů. Z nichž zajímavou volbou je možnost uložení vzhledu aplikace (angl. Save Windows layout). Tato volba zaručí, že rozmístění panelů a velikost panelů zůstane po příštím spuštění aplikace stejná jako po zavření aplikace. Hráč si tak může nastavit jednotlivé panely tak, jak mu to bude nejlépe vyhovovat, a toto nastavení si uchovat.

Volbu vypnutí či zapnutí zvuků je možné provést také kliknutím na ikonku na obr. 16. V obou případech to znamená, že zvuky všech herních objektů budou ztišeny. Tvůrci herních objektů však mohou toto nastavení obejít a přesto zvuky spouštět, pokud nevyužijí standardní aplikační rozhranní pro spouštění zvuků. Žádný oficiální herní objekt dodávaný s touto aplikací se tak však nechová.



Obr. 15. Nastavení aplikace



Obr. 16. Vypnutí a zapnutí zvuků

Výběr herních serverů do přihlašovacího dialogu v nastavení chybí proto, že pokud bude chtít uživatel používat jiný herní server, pak může spustit aplikaci, tak jak je uvedeno v odstavci 11.2. Případně je možné přepsat konfigurační soubor *Sokoban.exe.config*, který je uložen ve složce s nainstalovanou aplikací a dosáhnout tím přidání serveru do přihlašovacího dialogu.

11.7 Konzole

Konzole v hlavní aplikaci slouží pouze k zobrazování případných chybových informací, které se mohou při provozování aplikace vyskytnout.

Kapitola 12

12 Ladící nástroje

Tato kapitola popisuje některé způsoby ladění, které shledávám zajímavými a o kterých věřím, že by mohly být pro čtenáře užitečné.

12.1 DebuggerIX

DebuggerIX je třída, která slouží pro vypisování ladících informací v aplikaci.

12.1.1 Motivace pro vytvoření ladící třídy

Pro ladění aplikace jsem potřeboval umísťovat do kódu příkazy, které budou vypisovat informace o právě prováděné operaci, aby bylo možné dohledávat chyby. To je poměrně běžná praxe, nicméně v jazyce C# má tento přístup jednu velkou nevýhodu. Uvažme, že chceme například logovat průběh *for* cyklu s iterační proměnnou *i*. Příkaz by pak vypadal například takto:

```
DebuggerIX.WriteLine(“#: ”+i.ToString() +“|Val: ”+value.ToString());
```

Tento příkaz bude každou hodnotu proměnné *i* převádět na řetězec a pokaždé musí na haldě alokovat další paměť, podobně pro proměnnou *value*. Z tohoto důvodu by bylo hloupé nechávat podobné ladící příkazy v programu, protože bychom nutili garbage collector k častější práci a zbytečně bychom ztráceli výkon aplikace. Na druhou stranu ladění by se tím stávalo obtížnějším, protože bychom museli neustále odkomentovávat a zakomentovávat tyto příkazy.

Běžnou praxí je, že těla ladících funkcí se obalí preprocesorovými příkazy, které zabrání jejich kompilování a funkce jsou pak pouze prázdné schránky. Toto řešení by nám ovšem nepomohlo, protože jde o parametr funkce, jehož sestavení nás stojí výkon aplikace.

Řešením, které jsem použil ve své aplikaci, je označení všech ladících funkcí atributem: [Conditional("DEBUG")]. Toto je atribut, který říká, že funkce bude zkompilována pouze, pokud je nadefinována konstanta DEBUG, pokud tedy ve Visual Studiu vybereme jinou konfiguraci, která není určená pro ladění, tak veškeré funkce označené atributem uvedeným výše nebudou vůbec zkompilovány a tedy ani jejich volání nebudou zkompilována, což znamená, že můžeme bez obav umísťovat ladící výpisy v potřebném množství a nemusíme se obávat, že aplikace tím ztratí výkon.

12.1.2 Ladění v praxi

Ladění je rozděleno na jednotlivé typy zpráv. Při vkládání příkazů *DebuggerIX.WriteLine* do aplikace či herních objektů je možné označit každou tuto informaci štítkem z *DebbugerTag* výčtu:

```
public enum DebuggerTag
{
    Net,
    Game,
    PluginDraw,
    SimulationNotableEvents,
    SimulationProcessedEvents,
    XmlValidation,
    Plugins,
    AppComponents,
    Keyboard,
    AppParamsParsing
}
```

Výsledný příkaz pak vypadá takto:

```
DebuggerIX.WriteLine(DebuggerTag.Keyboard, "KeyIsDown", key);
```

Ladící informace, které se objeví ve výsledném logu lze nastavit pomocí parametrů aplikace *--log-exclude-tags* a *--log-permit-tags*, které zakáže dané tagy a zbytek povolí, resp. povolí dané tagy a zakáže ostatní.

Příklad, kdy aplikace uloží do složky *D:\Thesis\Log* soubor s názvem ve tvaru *log-<aktuální datum>.txt* a navíc bude aplikace logovat všechny informace, které nemají štítek *PluginDraw* a *Plugins*:

Sokoban.exe --enable-debuggin D:\Thesis\Log --log-exclude-tags PluginDraw Plugins

12.2 jQuery.dump a Firebug

Při ladění JavaScriptu se velmi často stává, že je potřeba si vypsat obsah objektu či pole. Do jisté míry je možné si vystačit s funkcí *alert(string)*, která zobrazí modální okno s požadovaným řetězcem. Bohužel tato metoda obsah objektů ani polí nezobrazí a pro komplikovanější ladění je naprosto nevhodná. Obecně ladění JavaScriptu bez přídavných ladících nástrojů je velmi zdlouhavé a zbytečně náročné.

Užitečným pomocníkem je plugin *jQuery.dump*, který obsahuje jedinou funkci *$.dump(param)*. Tato funkce zobrazí v novém okně kompletní obsah zadaného parametru. Zobrazení informací do nového okna je rozumné z toho důvodu, že JavaScript často mění strukturu HTML stránky, provádí animace apod., což může vést k tomu, že jednoduše nezbývá místo pro výpis ladících informací.

Dobrým pomocníkem pro ladění JavaScriptu je také rozšíření Firebug do webového prohlížeče Firefox, rozšíření poskytuje podobné možnosti ladění, jaké jsou známé programátorům v MS Visual Studiu, což ušetří mnoho času.

Ladící nástroje jsou pro programátora v JavaScriptu nezbytností a pakliže by vám nevyhovoval žádný z výše uvedených nástrojů, je vhodné si alespoň psát testovací kód.

Kapitola 13

13 Závěr

Cílem mé práce bylo vytvoření varianty hry Sokoban a komunitní webové aplikace pro hráče, která hráčům umožní srovnávat jejich dosažené výsledky. Hra se měla sestávat z dvou herních módů - pro jednoho a síťovou hru pro dva hráče. Dále mělo být možné vytvářet herní objekty formou pluginů. Pro vytváření kol měl být vytvořen editor a do aplikace měly být zapojeny existující řešitelé kol. Podle mého názoru bylo těchto cílů dosaženo.

13.1 Čeho bylo dosaženo?

Podařilo se vytvořit herní jádro, které svým fungováním dovoluje hrát hru pro jednoho hráče i hru pro hráče dva, aniž by musel tvůrce herních objektů speciálně svůj herní objekt připravovat na síťovou verzi hry. Toto vidím jako jeden z největších úspěchů mé práce, jelikož šlo o vytvoření hybridního systému, který se bude tvářit jako real-timový a zároveň bude jeho chování popsatelné pomocí jednotlivých kroků.

Řešitele kol se podařilo integrovat do aplikace a fungují bez problémů, jako nevýhodu spatřuji to, že někteří řešitelé jsou vázaní na aplikaci tvůrce řešitele a tak jejich používání může být pro uživatele trochu zvláštní. Tento problém však nelze řešit bez aktivní spoluúčasti ze strany tvůrců řešitelů.

Vytváření vlastních kol bylo zakomponováno do webové aplikace a dovoluje vytvářet jednoduše nová kola a to i pro různé varianty Sokobana. Díky své architektuře je možné velmi jednoduše přidávat nové varianty Sokobana do editoru.

13.2 Možnosti rozšíření a vylepšení

Tvorba mé práce se zdržela z důvodů vybrání nevhodné technologie na počátku, proto nebylo vytvořeno dostatečné množství herních objektů pro demonstraci možností pluginového systému. Například velmi jednoduše by šlo zrealizovat variantu s různobarevnými bednami a různobarevnými cílovými políčky. Díky předpřipravené třídě pro pohyb herních objektů je velmi jednoduché i vytvářet varianty s pohyblivými objekty s minimem vlastní práce. Vytváření variant provádět velmi snadno a je možné jich vytvořit velké množství a tím si i udržet hráče.

Herní objekty by dále mohly mít uživatelské nastavení. Nyní je možné objektům nastavovat vlastnosti pomocí XML v zápisu kola, avšak toto není vhodné pro uživatelské nastavení, jejímiž požadavky může být vypnutí animace herního objektu apod.

Editor kol by bylo možné vylepšit o funkci import, která by ze znakového zápisu kola dokázala načíst kolo do editoru a umožnila jej dále editovat. Fungovala by na principu mapování jednotlivých znaků na herní objekty, tedy by tato funkce dále rozšiřovala schéma varianty pro editor. Import by byl užitečný proto, že by bylo možné snadno využít existující zápisy kol, které jsou dostupné na Internetu.

Největším vylepšením aplikace by bylo přepsání kritických částí aplikace do nativního zdrojového kódu, čímž by se mohlo dosáhnout zvýšení výkonu aplikace a ušetření zdrojů počítače. Tím by se vyřešil fakt, že .NET není pro programování her příliš vhodný.

Literatura

[1] Microsoft Corporation. MSDN - XNA Game Studio 3.1. [Online]. http://msdn.microsoft.com/en-us/library/bb196414.aspx#ID4ELF

[2] Alex Calvo. (2004, Feb.) MSDN Magazine. [Online]. http://msdn.microsoft.com/en-us/magazine/cc164015.aspx

[3] Microsoft Corporation. (2008, Jan.) XNA Creators Club Online. [Online]. http://creators.xna.com/en-US/sample/winforms_series2

[4] Ian Griffiths Chris Sells, *Programming WPF, Second Edition.*: O'Reilly Media, December 2008.

[5] Adolf Marinucci. (2010, July) AvalonDock. [Online]. http://avalondock.codeplex.com/

[6] Sokoban Project. Sokoban Project. [Online]. http://sokobano.de/wiki/index.php?title=Links#Sokoban_Solvers

[7] Joris Wit. Sokoban++ Solvers. [Online]. http://www.joriswit.nl/sokoban/en/solvers.htm

[8] Joris Wit. Sokoban++ Solver SDK. [Online]. http://www.joriswit.nl/sokoban/files/games/devkit.zip

[9] Go yong. (2010, July) BoxSearch -- My Sokoban Solver. [Online]. http://notabdc.vip.sina.com/EngPage/e_solver.htm#BoxSearch

[10] Brian Damgaard. (2010, July) Sokoban YASC. [Online]. http://sourceforge.net/projects/sokobanyasc/files/

[11] David White. (2006, Mar.) PocoSolv. [Online]. http://home.pacific.net.au/~davidw/pocosolv/

[12] University of Alberta. (2004, Mar.) Rolling Stone. [Online]. http://webdocs.cs.ualberta.ca/~games/Sokoban/program.html

[13] (2008, 28) Wintellect's Power Collections for.NET. [Online]. http://powercollections.codeplex.com/

[14] Sacha Barber. (2008, May) CODE PROJECT. [Online]. http://www.codeproject.com/KB/dotnet/AddInModel.aspx

[15] Microsoft Corporation. MSDN - Application Domains Overview.

[Online]. http://msdn.microsoft.com/en-us/library/2bh4z9hs(VS.71).aspx

[16] Mehdi Achour et al. PHP Manual. [Online]. http://cz.php.net/manual/en/history.php.php

[17] MySQL AB. MySQL 5.1 Reference Manual. [Online]. http://dev.mysql.com/doc/refman/5.1/en/differences-from-ansi.html

[18] Petr Daněk. Root. [Online]. http://www.root.cz/clanky/velky-test-php-frameworku-2008/

[19] avnetlabs.com. avnetlabs. [Online]. http://avnetlabs.com/php/php-framework-comparison-benchmarks

[20] David Grudl. (2009) Root. [Online]. http://zdrojak.root.cz/serialy/zaciname-s-nette-framework/

[21] W3C DOM IG. (1995) W3C. [Online]. http://www.w3.org/DOM/

[22] John Resig. (2005, Sep.) ejohn.org. [Online]. http://ejohn.org/blog/flexible-javascript-events/

[23] Peter-Paul Koch. quirksmode.org. [Online]. http://www.quirksmode.org/js/this.html

[24] JSON. JSON. [Online]. http://json.org

[25] json.org. (2010, July) Introducing JSON. [Online]. http://json.org

[26] James Clark and Steve DeRose. (1999, Nov.) W3C. [Online]. http://www.w3.org/TR/xpath/

[27] Martin Všetička. SoTh. [Online]. http://sokoban.research-site.net

[28] (2010, July) Moje IP. [Online]. http://www.mojeip.cz/

[29] Microsoft Corporation. (2008, Jan.) XNA Creators. [Online]. http://creators.xna.com/en-US/sample/winforms_series1

[30] PHP Frameworks. PHP Frameworks. [Online]. http://www.phpframeworks.com/top-10-php-frameworks/

[31] Jim Mischel. (2007, June) DevSource. [Online]. http://www.devsource.com/c/a/Languages/A-Priority-Queue-Implementation-in-C/

Příloha A

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns="http://www.martinvseticka.eu/SoTh"
           targetNamespace="http://www.martinvseticka.eu/SoTh"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

  <!-- Game objects -->
  <xs:element name="GameObject" abstract="true" type="GameObjectType" />

  <xs:complexType name="GameObjectType">
    <xs:sequence>
      <xs:element name="Version" type="xs:string" minOccurs="0"
                  maxOccurs="1" />
      <xs:element name="PosX" type="xs:integer" />
      <xs:element name="PosY" type="xs:integer" />
      <xs:element name="InitializeMessage" type="xs:string"
                  minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="Dimensions">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Width" type="xs:integer" />
        <xs:element name="Height" type="xs:integer" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="GameObjects">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="GameObject" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- One game round-->
  <xs:element name="Round">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1"
                    maxOccurs="1" />
        <xs:element ref="Dimensions" />
        <xs:element name="Variant" type="xs:string" minOccurs="1"
                    maxOccurs="1" />
        <xs:element ref="GameObjects" minOccurs="1" maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Quest (also known as League) -->
  <xs:element name="SokobanQuest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string" minOccurs="1"
                    maxOccurs="1"/>
```

```
        <xs:element ref="Round" minOccurs="1" maxOccurs="unbounded" />
        <xs:element name="NextQuest" type="xs:string" minOccurs="0"
                    maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```