VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe
Individual Professional Practice in the Company

2017

Radek Megyesi

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Radek Megyesi**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: EDIacademy, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **RNDr. Eliška Ochodková, Ph.D.**

Konzultant bakalářské práce: Ing. Lukáš Domin

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Čestné prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 28. 4. 2017

.....

podpis

EDIacademy, s.r.o.

Frýdecká 333
737 01 Český Těšín

Vážený pán

Radek Megyesi
Františka Formana 237/31
Ostrava – Dubina 700 30

Souhlas se zveřejněním bakalářské práce

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Českém Těšíně dne 10. dubna 2017

Ing. Rostislav Zabystrzan
jednatel

Poděkování

Rád bych zde poděkoval Ing. Jakubovi Suchankovi za možnost absolvovat individuální odbornou praxi ve společnosti EDIacademy s.r.o. Dále mému mentorovi Ing. Lukášovi Dominovi za ochotu, vstřícnost a předané zkušenosti v průběhu absolvování odborné praxe. V neposlední řadě bych rád podělkoval RNDr. Elišce Ochodkové, Ph.D. za cenné rady při tvorbě této práce.

Abstrakt

Cílem této bakalářské práce je popsat průběh individuální odborné praxe, kterou jsem absolvoval ve společnosti EDIacademy s.r.o. První část práce se zabývá představením společnosti, popisem mého pracovního zařazení, obecným představením aplikace a popisem použitých technologií. Další část práce tvoří popis mých pracovních úkolů společně s jejich řešením, které jsem použil. V závěru práce jsou popsány zkušenosti získané během studia i praxe a zhodnocení dosažených výsledků.

Klíčová slova

JavaScript, HTML, CSS, Quintus, jQuery, JSON, EDIacademy, webová aplikace, AJAX, herní jádro, SVN, Subversion

Abstract

The aim of this bachelor thesis is to describe the process of individual professional practice, which I attended in the company named EDIacademy s.r.o. First part of this work deals with presentation of company, description of my work position, general presentation of application and description of used technologies. Next part of work consists of description of my work tasks along with their solutions that I used. At the end of this work are described experiences gained during study and practice and evaluation of achieved results.

Key words

JavaScript, HTML, CSS, Quintus, jQuery, JSON, EDIacademy, web application, AJAX, game engine, SVN, Subversion

Obsah

Seznam použitých zkratek a symbolů

HTML	-	HyperText Markup Language
CSS	-	Cascading Style Sheets
JSON	-	JavaScript Object Notation
AJAX	-	Asynchronous JavaScript and XML
XML	-	Extensible Markup Language
JPG (JPEG)	-	Joint Picture Experts Group.
SVN	-	Subversion
FTP	-	File Transfer Protocol
OOP	-	Object Oriented Programming (objektově orientované programování)
IDE	-	Integrated Development Environment

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Jako bakalářskou práci jsem si zvolil absolvování individuální odborné praxe ve firmě EDIacademy s.r.o., jelikož jsem již měl základní zkušenosti s HTML a CSS a chtěl jsem si také prohloubit mé znalosti z oblasti programování v JavaScriptu, se kterým jsem se do té doby setkal pouze okrajově. Zaujala mě možnost pracovat na vývoji webové aplikace určené pro zlepšování znalostí žáků základních a středních škol, implementované pomocí nadstavby programovacího jazyka JavaScript, která se nazývá Quintus.

1.1 Představení společnosti

Společnost EDIacademy s.r.o [1]. vznikla za účelem podpořit matematické a kritické myšlení žáků základních škol a posléze i absolventů škol středních a zahájila práci na vývoji didaktických elektronických materiálů pro výuku matematiky. Společnost se při analýze dostupných možností seznámila s koncepcí prof. Hejného a v srpnu 2015 započala spolupráci se společností H-mat, o.p.s. [2].

EDIacademy s.r.o [1]. dlouhodobě spolupracuje s Ostravskou Univerzitou a Univerzitou Komenského v Bratislavě, kde za spolupráce s fakultními základními školami provádí výzkum a testování nově vznikajících elektronických výukových pomůcek.

Základním cílem společnosti je, aby žák na konci základního vzdělávání:

- Rozpoznal a pochopil problém
- Promyslel a naplánoval způsob řešení
- Využíval získané vědomosti a dovednosti k objevování různých variant řešení
- Samostatně řešil problémy
- Užíval při řešení problémů logické, matematické a empirické postupy
- Ověřoval prakticky správnost řešení problémů
- Kriticky myslel

Firma vyvíjí webové aplikace s použitím HTML5 a JavaScriptu (převážně framework Phaser).

Obrázek 1: Logo společnosti

1.2 Pracovní zařazení

Ve firmě jsem se podílel na vývoji webové aplikace jako programátor. V průběhu praxe mi byly zadávány jednotlivé požadavky na to, jak má aplikace vypadat, jak se má chovat a fungovat. Podle těchto pokynů jsem aplikaci vytvářel a následně rozšiřoval. Na aplikaci jsem pracoval převážně z hlediska aplikační logiky. Z pohledu grafiky jsem vytvářel povětšinou stylování pomocí CSS. Hotové grafické prvky (obrázky), které jsou použity v aplikaci, pak byly vytvořeny firemním grafikem.

1.3 Představení aplikace

Aplikace, na které jsem pracoval, je webová aplikace navržena výše představenou společností EDIacademy s.r.o., a má být určena k výuce a procvičování matematických operací týkajících se sčítání a odečítání zlomků. Aplikace tedy představuje prostředí, ve kterém je možné navrhnout početní příklady se zlomky, které je následně potřeba vyřešit. Vše je podáno zábavnou formou hry, ve které číšníci vycházející od baru roznáší kávu hostům, kteří sedí na kobercích. Množství kávy je reprezentováno pomocí zlomků. Směr pohybu číšníků může být ovlivněn směrovými dlaždicemi a popřípadě omezen různými druhy překážek. Rozložení objektů na hrací ploše představující kavárnu a celkové nastavení aplikace může být specifikováno předem pomocí možnosti ukládání a načítání úlohy.

1.4 Použité technologie

V aplikaci jsou použity známé technologie z oblasti pro vývoj webových aplikací, které jsou popsány v následujícím texu.

1. **HTML** - jazyk používaný pro tvorbu webových stránek a jejího obsahu [7]

Tento jazyk používá specifických značek, tzv. tagů, pomocí kterých je tvořena struktura webové stránky i s jejím obsahem. Tuto vytvořenou strukturu je pak možné dále upravovat a definovat její vzhled pomocí dalších jazyků a nástrojů.

V aplikaci je použito HTML pro zobrazení některých částí obsahu, konkrétně verze 5 [8, 9].

2. **CSS** – jazyk, jenž popisuje způsob, jakým jsou zobrazovány HTML nebo XML dokumenty [10]

Jazyk má mnoho různých parametrů definující vzhled a chování. Tyto parametry se pak přidělují a nastavují specifickým identifikátorům, které odkazují na jednotlivé prvky v dokumentu.

Pro úpravu grafické podoby některých komponent v aplikaci je použito klasické CSS verze 3.

3. **JavaScript –** programovací jazyk používaný převážně pro tvorbu webových stránek a aplikací [11]

Jazyk se ve velké většině používá v oblasti webů, avšak využívá se i pro vývoj desktopových a jiných aplikací.

Aplikační logika je vypracována právě pomocí jazyka JavaScript, který je použit například k zachycování událostí, vytváření dynamického obsahu nebo načítání souborů. Tento jazyk je zde užit jak v jeho základní podobě, tak i v podobě využití knihovny jQuery a herního jádra Quintus.

4. **Quintus** – herní jádro na bázi HTML a JavaScriptu, které bylo vyvinuto pro vytváření her a aplikací pro počítače, mobilní přístroje a jiná zařízení [3]

Aplikace je z velké části vytvořena právě pomocí Quintusu a rozšiřována pak dále pomocí dalších zde zmíněných technologií. Quintus poskytuje aplikaci základní chování a mechanismy, které jsou důležité pro hlavní koncept celé aplikace.

5. **jQuery** – je knihovna napsaná v jazyce JavaScript [12]

Tato knihovna sestává z velké řady funkcí, které mají za úkol usnadnit práci s jazykem JavaScript.

2 Úkoly zadané během odborné praxe

2.1 Příprava prostředí pro vývoj

Zadání

Můj první úkol se týkal konfigurace a přípravy vývojového prostředí (tzv. IDE) v počítači a nastavení dalších prerekvizit, které byly nezbytné pro budoucí úkoly a vývoj aplikace. Konkrétně se jednalo o instalaci vývojového prostředí, ve kterém jsem pak po zbytek praxe pracoval. Dále šlo o nastavení a připojení se k serveru, na kterém se nacházel systém SVN určený pro verzování a správu zdrojových kódů aplikace a jejích dalších součástí (souborů). Na tento server jsem postupně nahrával všechny následné verze aplikace při vývoji.

Řešení

Nainstaloval jsem doporučené vývojové prostředí do počítače, konkrétně se jednalo o aplikaci NetBeans, která podporuje práci s mnoha známými programovacími jazyky. Dále jsem si rozšířil prohlížeč o vylepšení umožňující propojení vývojového prostředí s prohlížečem, a tím jednodušší zobrazování a ladění samotné aplikace. Vývojové prostředí NetBeans tak při každém spuštění aplikace spustilo lokální server (pokud už se tak nestalo). Aplikace tak mohla být na tomto lokálním serveru spuštěna a laděna.

Dále jsem ve vývojovém prostředí nastavil všechny náležitosti potřebné k připojení na server SVN. Toto nastavení tak umožnilo jednoduchou správu verzí aplikace a ukládání samotných verzí na firemní server bez nutnosti použití dalších programů.

Přibližný počet dnů strávených řešením tohoto zadání: 1

2.2 Seznámení s herním jádrem Quintus

Zadání

Nyní, když byl počítač připraven pro vývoj aplikace, bylo mým úkolem seznámit se s herním jádrem Quintus, prostudovat dokumentaci a následně pomocí získaných znalostí rozšířit jednoduchou aplikaci o další úrovně, herní a ovládací prvky, grafiku, animace a jiná vylepšení. Cílem bylo procvičit si základní práci s herním jádrem a jeho vlastnosti.

Řešení

Po přečtení a prostudování základní dokumentace [4], jsem si lokálně zprovoznil aplikaci, která je poskytována jako ukázka pro herní jádro. Jednalo se o jednoduchou 2D plošinovou hru, která obsahovala pouze jednu krátkou úroveň. V této úrovni bylo za úkol dostat se s objektem (představujícím hráče) skrze omezení a nástrahy z bodu A do bodu B. Tuto aplikaci jsem následně rozšířil o další úroveň, která obsahovala různé bonusy, nástrahy, překážky a jiná vylepšení.

Přibližný počet dnů strávených řešením tohoto zadání: 2

2.3 Samotná aplikace

V následující části budu popisovat úkoly, které vedly k postupnému vytvoření výsledné aplikace.

2.3.1 Základní kostra

Zadání

Poté, co jsem si osahal ty nejzákladnější části vývoje aplikace pomocí herního jádra Quintus [3], bylo mým dalším zadáním vytvořit základní kostru celé aplikace, která se tak mohla dále rozšiřovat a vylepšovat. Do projektu jsem měl vytvořit provizorní pozadí podle zadaných parametrů, implementovat pohyb kamery nad plochou aplikace pomocí šipek na klávesnici a vytvořit nový typ objektu, který byl základní definicí pro další později vytvořené objekty. Pomocí tohoto objektu jsem pak měl vytvořit hrací plochu aplikace. Posledním požadavkem bylo, aby se každý čtverec hrací plochy rozsvítil (změnil barvu) pokud se nad ním nachází kurzor myši, aby tak bylo zřejmé, se kterým dílkem hrací plochy bude prováděna interakce.

Řešení

Podle zadaných rozměrů jsem si pomocí grafického programu GIMP [13] vytvořil obrázek ve formátu JPG s motivem šachovnice, který jsem pak použil jako pozadí pro hrací plochu celé aplikace.

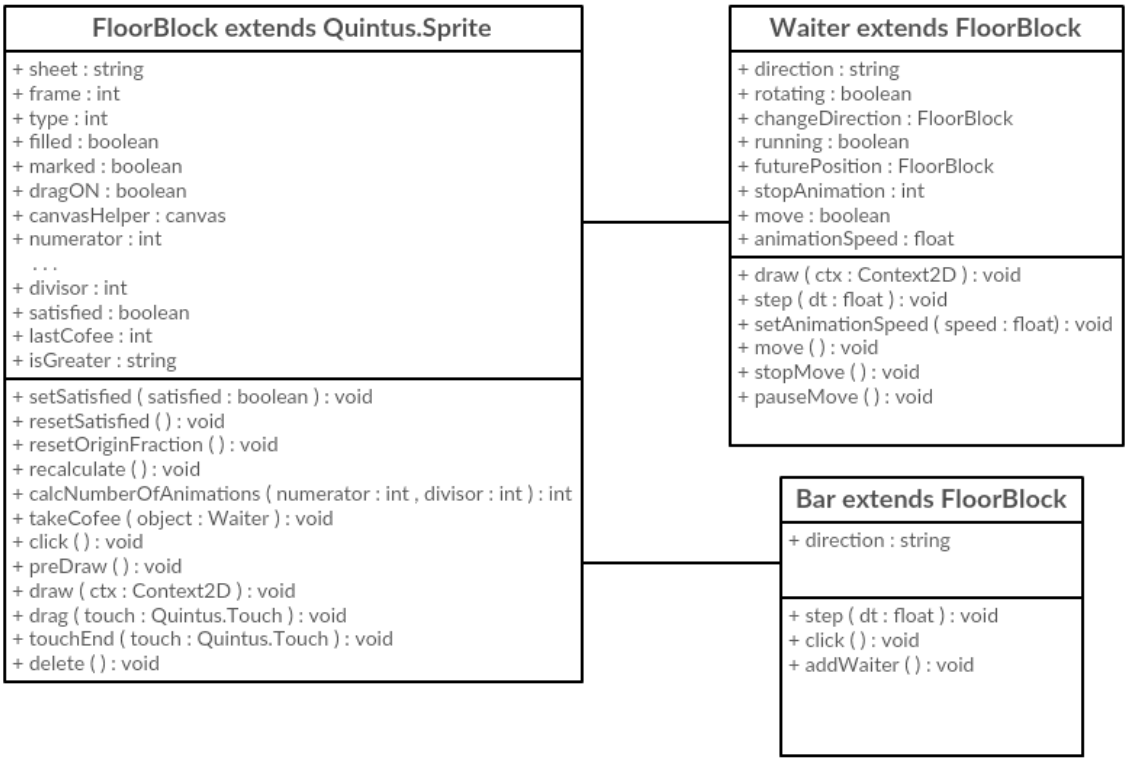
Dále jsem naimplementoval pohyb kamerou. Quintus [3] umožňoval použití a zachytávání příkazů ze vstupního zařízení (v našem případě klávesnice), ale pro pohyb kamery zde žádné předdefinované funkce nebyly. Proto jsem vytvořil neviditelný objekt, který nemá žádné rozměry a vložil jej do středu aplikace. Tento objekt následně reaguje na podněty z klávesnice a rozpohybuje se v daném vertikálním a horizontálním směru podle stisknutých směrových kláves. Výřez okna aplikace pak sleduje tento objekt a tím je zajištěn pohyb kamery na zobrazovacím zařízení.

Jako další krok jsem vytvářel nový typ objektu, kterému jsem nastavil základní vlastnosti a chování, jako jsou například velikost, zda je objekt průchozí anebo zda je objekt možno přesouvat. Tyto vlastnosti se pak dále liší v závislosti na typu. Uprostřed pozadí jsem z tohoto objektu o rozměru jednoho čtverce šachovnice na pozadí vytvořil síť sloužící jako hrací pole. Požadavkem bylo, aby se rozměr tohoto objektu i hrací plochy dal měnit dynamicky v závislosti na předdefinovaných globálních parametrech aplikace. Toto v případě velikosti objektu nebyl problém. V případě hrací plochy jsem však musel vytvořit funkci, která dynamicky vytvoří pole těchto objektů představujících dlaždice, a to podle zadaného počtu řádků a sloupců. Tyto počty odpovídají vyobrazeným dlaždicím na obrázku pozadí hrací plochy.

Nakonec jsem doimplementoval požadované zvýraznění dílku hrací plochy při přítomnosti kurzoru výměnou obrázku za jiný a zpět (po opuštění dílku kurzorem). Na obrázku č. 2 je zobrazena část hrací plochy se zvýrazněným dílkem, nad kterým se nachází kurzor.

Obrázek 2: Změna vzhledu dlaždice při najetí kurzorem

V obrázku č. 3 je zobrazen diagram tříd, který obsahuje třídu „FloorBlock“ pro obecný objekt na hrací ploše, třídu „Waiter“ pro číšníky a třídu „Bar“ pro objekt, ze kterého číšníci na hrací ploše vycházejí.



Obrázek 3: Třídní diagram - základní typy objektů

Přibližný počet dnů strávených řešením tohoto zadání: 2

2.3.2 Paleta s objekty a ovládacími prvky

Zadání

Dalším zadáním bylo přidání HTML elementu do aplikace. Tento element, nazývaný paleta, má obsahovat použitelné objekty a také ovládací prvky webové aplikace. Tyto objekty musí být možné přesouvat z palety na hrací plochu pomocí principu „drag and drop“ (táhni a pusť) a do palety musí být generovány dynamicky, a to z důvodu možnosti jednoduššího budoucího rozšíření o další objekty.

Řešení

Vytvořil jsem do stránky HTML element, který jsem nastyloval podle zadaných parametrů pomocí CSS. Nahrávání objektů do palety (obrázků představující objekty) jsem zrealizoval pomocí JavaScriptové knihovny jQuery [12] a cyklu „for-each“. Každý obrázek je tak přidán do palety dynamicky na základě specifikovaného výčtu obrázků, které jsou pak postupně jeden po druhém vkládány do palety. I zde jsem si vytvořil dočasné obrázky s názvy objektů, které byly později nahrazeny. Výsledná podoba palety je vyobrazena na obrázku č. 4, kde je již použita finální grafika dodána firmou.

Nejtěžší část tohoto úkolu nastala, když jsem měl implementovat přesouvání objektů z palety do hracího pole. Pro chycení obrázku myší a jeho následného puštění jsem využil funkce knihovny jQuery [12], které realizovaly toto chování [14, 15]. Chycení objektu v paletě funguje následovně: posluchač nastavený na objektech reaguje na stisk myši uživatelem a obrázek tohoto objektu je nakopírován ke kurzoru myši a udržován, dokud nedoje k puštění objektu. Problém však nastal v okamžiku, kdy se měl objekt puštěný nad hrací plochou umístit do pole. Jelikož souřadnice plátna (canvasu) prohlížeče pro vykreslování byly rozdílné od těch, jenž používalo plátno pro vykreslování v Quintusu [3], musel jsem vytvořit funkci pro předávání a přepočet souřadnic objektu mezi plátny. Další funkce, kterou jsem implementoval, následně vytvoří v hracím poli na daném místě objekt, který je shodný s přesouvaným objektem z palety. Tomuto nově vytvořenému objektu jsou ihned nastaveny všechny potřebné parametry.

Výpis zdrojového kódu č. 1 zobrazuje implementaci vkládání objektů na hrací plochu pomocí využití funkce *droppable()* [15] z knihovny jQuery [12] a vytvořením funkce do události *drop*. V této vytvořené funkci jsem použil již zmíněnou funkci pro přepočet souřadnic *getObject()*, která vrací objekt nacházející se na zadaných souřadnicích v dané vrstvě aplikace. Objekt je pak na místo vložen a jsou mu ihned nastaveny požadované parametry podle jeho typu.

```
$("canvas").droppable({
    drop: function (event, ui) {
        var tileOfFloor = getObject(event.pageX, event.pageY, 0);
        if (tileOfFloor !== null && tileOfFloor.p) {
            var draggedObj = Task.getSettingByName(ui.draggable[0].id);
            if (tileOfFloor.p.filled === true) {
                var objectOnFloor = getObject(event.pageX, event.pageY, 1);
                Task.computeCount(objectOnFloor.p.sheet, +1);
                objectOnFloor.destroy();
            }

            if (ui.draggable[0].id === "kavamat") {
                Q.stage(1).insert(new Q.FB_Kavamat({
                    x: tileOfFloor.p.x,
                    y: tileOfFloor.p.y,
                    sheet: draggedObj.tile,
                    frame: 0,
                    dragON: draggedObj.moveable,
                    numerator: 0,
                    divisor: 1,
                    direction: "up"
                }));
                Task.computeCount(ui.draggable[0].id, -1);
            }
```

```
            else {
                Q.stage(1).insert(new Q.FloorBlock({
                    i: 0,
                    x: tileOfFloor.p.x,
                    y: tileOfFloor.p.y,
                    sheet: draggedObj.tile,
                    frame: 0,
                    dragON: draggedObj.moveable
                }));
                Task.computeCount(ui.draggable[0].id, -1);
            }
            tileOfFloor.p.filled = true;
            tileOfFloor.p.dragON = draggedObj.moveable;
        }
    }
});
```

Zdrojový kód 1: Vkládání objektů na hrací plochu

Obrázek 4: Paleta s objekty

Popis palety začínající zleva nahoře:

bar – objekt, ze kterého vycházejí číšníci směrem z otevřené strany baru

koberec – objekt představující zákazníky s objednanou kávou (postavy jsou na koberec náhodně generovány v momentu položení koberce na hrací plochu)

směrové šipky – dlaždice měnící směr pohybu číšníka po hrací ploše

překážky – neprůchozí objekty, o které se číšník zastaví

tlačítko start – spustí simulaci roznášení kávy, po spuštění se tlačítko změní na „STOP“, které zastaví simulaci a uvede všechny objekty do stavu před spuštěním

tlačítko zrychlení – přepíná rychlost číšníka

tlačítko nastavení – zobrazí formulář s nastavením aplikace

U objektů, které je možno umístit na hrací plochu, se v jeho levé spodní části zobrazuje hodnota znázorňující počet, kolikrát je možno daný objekt použít.

Přibližný počet dnů strávených řešením tohoto zadání: 2

2.3.3 Označování a přesouvání objektů

Zadání

Realizovat označování objektů na hrací ploše (vizualizace označení), implementovat přesouvání objektů kurzorem myši po hrací ploše i případné vyměňování objektů mezi sebou, pokud to mají povoleny. Každý objekt tak měl mít parametrizováno, zda je možné ho přesouvat nebo nikoliv. Požadavkem bylo připravit si tyhle parametry tak, aby se daly za běhu aplikace také měnit. Poslední částí úkolu bylo zrealizovat mazání objektů z plochy jejich přetažením do oblasti palety.

Řešení

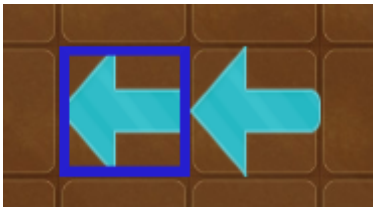
Nejprve jsem označování objektu vyřešil zaměňováním obrázků, kdy jsem si pro každý obrázek objektu vytvořil jeho kopii, avšak s již barevným rámečkem znázorňující označení. Toto řešení, jak se později ukázalo, nebylo dostačující a efektivní. V případě rozšiřování aplikace o další objekty, by zde byla nutnost vytvářet duplicitní obrázky. Tímto by také docházelo k neefektivnímu využití místa na disku.

Toto označování jsem nakonec zrealizoval objektem, který graficky vypadal jako rámeček a tento rámeček je poté zobrazován ve vrstvě nad objektem, pokud uživatel na tento objekt klikne. Názorná ukázka označeného a neoznačeného objektu je zobrazena na obrázku č. 5.

Pro přesouvání objektů po ploše má již Quintus [3] předpřipravené metody. Tyto metody nemají implementovanou logiku přesouvání, avšak dokážou být volány ve chvíli, kdy se objekt začne přesouvat, v našem případě uživatelem. Tímto způsobem pak bylo možné naprogramovat chování objektů pro přesun po hrací ploše.

Nejtěžší na tomto úkolu byla část, kdy jsem musel vymyslet algoritmus, který má za úkol kontrolovat několik situací, jenž mohou nastat. Algoritmus kontroluje, nad kterým dílkem hrací plochy byl objekt puštěn. Dále zda je možné ho na tento dílek umístit, zda nebyl upuštěn mimo hrací plochu a zda objekty prohodit, pokud se již na tomto místě nachází objekt s možností přesunu.

Nakonec jsem implementoval mazání objektů. Tyto objekty jsou smazány pouze v případě přetažení do oblasti palety.



Obrázek 5: Označený a neoznačený objekt

Přibližný počet dnů strávených řešením tohoto zadání: 4

2.3.4 Nezávislost aplikace na internetu

Zadání

Nakonfigurovat aplikaci tak, aby byla nezávislá na Internetu.

Řešení

Pro to, aby byla aplikace schopná fungovat i v případě nepřístupnosti k Internetu, musel jsem všechny závislosti uložit fyzicky k projektu. Vytvořil jsem si ve struktuře aplikace adresář, do kterého jsem tyto závislosti (knihovny funkcí) vložil a nakonfiguroval aplikaci tak, aby pracovala s těmito zdroji a neodkazovala se na knihovny funkcí na Internetu.

Přibližný počet dnů strávených řešením tohoto zadání: 0,5

2.3.5 Možnost nastavování hodnot pro číšníky a koberce

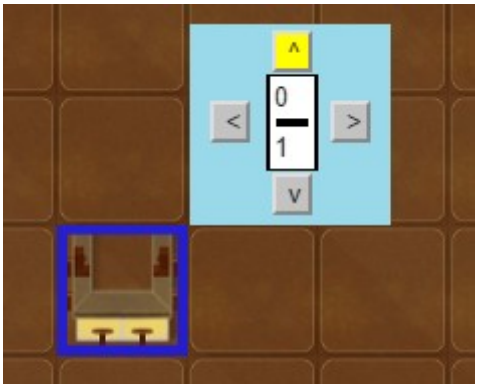
Zadání

Vytvořit ovládací HTML prvek, jenž se bude zobrazovat při kliku na objekt „bar“ a „koberec“. Pomocí tohoto ovládacího prvku bude možné nastavovat hodnotu kávy (zlomek), kterou číšník ponese a směr kudy se číšník po startu simulace vydá. V případě koberce se jedná pouze o množství kávy, kterou si zákazníci objednali. Požadavkem bylo implementovat kontrolu a omezení čísel, která je možno vložit do čitatele a jmenovatele zlomku na čísla od 0 do 99. Směr měl být proveden pomocí tlačítek, které také měly otáčet objekt kolem své osy. Všechny zadané parametry bylo pak třeba ukládat do objektu.

Řešení

Pomocí JavaScriptu [11] jsem vytvořil funkci, která měla za úkol zobrazovat rámec umožňující nastavování daného objektu („baru“ a „koberce“). V této funkci jsem vytvořil HTML element obsahující potřebné ovládací prvky, který se následně pomocí volání metody při kliku na objekt dynamicky vykresluje do stránky, konkrétně v pravé horní části od označeného objektu. Při odznačení zase zmizí. Tento dynamicky vytvořený a vložený prvek obsahuje 2 vstupní pole pro zadání zlomku reprezentující čitatele a jmenovatele. Pro nastavení „baru“ ještě navíc obsahuje 4 tlačítka směru. Při zobrazování jsou vždy načteny uložené parametry daného objektu. Výsledný ovládací prvek pro nastavování parametrů objetu je vyobrazen na obrázku č. 6.

Menší problém nastal, když jsem měl pro zadané hodnoty do zlomku omezit zadávání pouze na čísla, a to v rozmezí 0 až 99, s tím že do jmenovatele navíc není možné zadat nulu. Jelikož HTML vstupní formulářové prvky pro vkládání neposkytovaly dostatečnou možnost nastavení omezení, byl jsem nucen najít jiné řešení kontroly vstupů. Nakonec jsem se rozhodl použít regulární výraz (zkráceně regex či regexp). Tento specifický řetězec znaků představuje předpis (neboli masku), pomocí kterého jsme schopni určit, zda zadaný vstup odpovídá tomuto předpisu, či nikoli. Tuto filtraci jsem následně aplikoval na událost při stisku klávesy v zadávacím poli (změny hodnoty) a při jeho opuštění. Pokud se tedy uživatel pokusí zadat cokoli jiného než čísla v daném rozsahu, program tuto událost zachytí a tento znak zapsat nepovolí. Při zavření nastavovacího formuláře objektu (odznačením), jsou zadané parametry uloženy do objektu, jemuž náleží.



Obrázek 6: Nastavení objektu

Přibližný počet dnů strávených řešením tohoto zadání: 3

2.3.6 Inicializace číšníků a implementace pohybu

Zadání

Přidat do palety s objekty tlačítko „START“, určující běžící nebo zastavený stav aplikace. Stisknutí tlačítka vloží číšníky na hrací plochu hned vedle baru určeným směrem, nastaví jim potřebné hodnoty zlomku převzaté z baru, ze kterého vychází a uvede je do pohybu. Opětovné stisknutí tohoto tlačítka při běžícím stavu má způsobit zastavení simulace a obnovení aplikace do stavu před startem.

Dále do levého horního rohu zobrazovat stav aplikace v grafické podobě. Pro zastavený stav červený čtverec, pro běžící zelený trojúhelník.

Řešení

Vytvořil jsem HTML ovládací prvek reprezentující tlačítko „START“ a vložil jej do palety. Pro objekt reprezentující bar, jsem vytvořil metodu, která zajišťovala vložení číšníků (pro každý bar vždy pouze jednoho) na hrací plochu. Tato metoda kontroluje, zda je možné číšníka v daném směru na plochu vložit. Pokud se v daném směru v poli hned vedle baru nenachází překážka, je vytvořená instance objektu reprezentujícího číšníka a vložena na plochu (obrázek č. 8). Této instanci jsou předány informace udávající směr pohybu a zlomek reprezentující kolik ponese kávy.

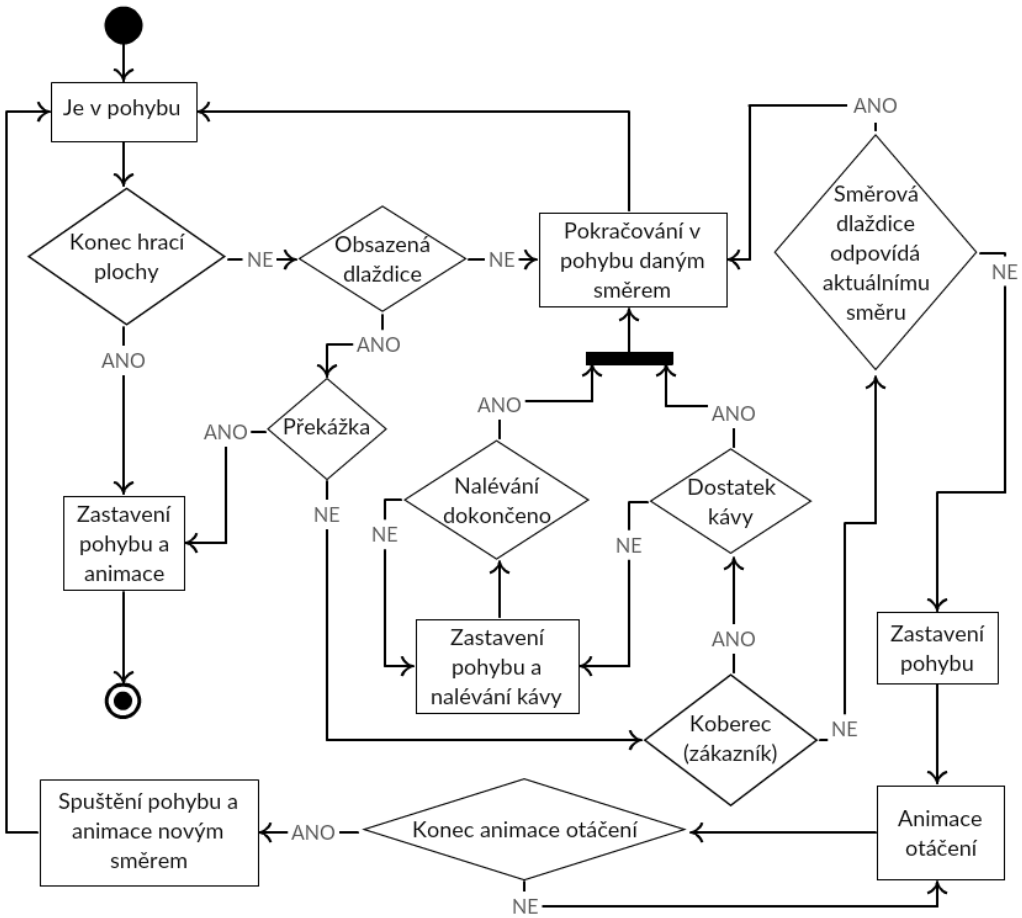
Nyní, když jsem naimplementoval vkládání číšníků na hrací plochu, bylo třeba je uvést do pohybu. K tomuto jsem využil metodu *step()* [6], kterou obsahují objekty Quintusu [3], pokud jsou vytvořeny pomocí třídy *Sprite*. Tato metoda je pravidelně v určitém (velmi krátkém) časovém intervalu volána a má za úkol aktualizovat atributy objektu. Tuto metodu je možné mnoha způsoby rozšiřovat. Do této funkce jsem tedy vytvořil algoritmus, který zajišťuje posouvání číšníka. Při každém volání této metody *step()* [6], se číšník posune o daný počet pixelů v jeho určeném směru po hrací ploše. Ve výsledku se tak číšník plynule pohybuje. Životní cyklus číšníka je znázorněn na obrázku č. 7 pomocí stavového diagramu.

Do metody *step()* [6] je později implementováno další chování, které budu popisovat v následujících úkolech.

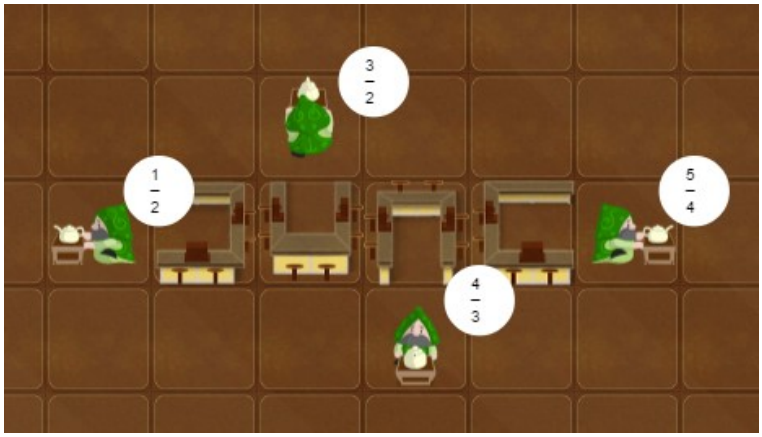
Tento algoritmus provádí následující funkce:

- Posouvá číšníka horizontálním nebo vertikálním směrem podle zadaných parametrů
- Přehrává animace pohybu
- Mění rychlost pohybu číšníka a jeho animace v závislosti na nastavené rychlosti přehrávání
- Kontroluje budoucí pozici a podle toho co se v místě budoucího přesunu nachází, pak:
    - v případě překážky nebo konce hrací plochy zastaví
    - v případě směrové šipky podle ní změní svůj směr pohybu
    - v případě zákazníka (koberce) začne rozlévat kávu podle požadavků daných zákazníkem

Pro zobrazování stavu aplikace jsem využil podobný princip. Vytvořil jsem si další objekt, který jsem vložil do levého horního rohu aplikace. V jeho metodě *step()* [6] je kontrolováno, zda jsou zastaveni všichni číšníci či nikoliv. Podle toho pak objekt zobrazuje ikonu znázorňující běžící nebo zastavený stav aplikace a mění text v tlačítku start a jeho chování při stisku.



Obrázek 7: Stavový diagram životního cyklu číšníka v aplikaci



Obrázek 8: Inicializovaní číšníci

Přibližný počet dnů strávených řešením tohoto zadání: 3

2.3.7 Směrové dlaždice

Zadání

Měnit směr pohybujícího se číšníka podle směrové dlaždice, na kterou stoupl.

Řešení

Implementaci tohoto řešení jsem realizoval taktéž ve výše zmíněné metodě *step()* [6] číšníka. Před každým posunutím číšníka, je v metodě provedena kontrola následující dlaždice hrací plochy, na kterou má číšník namířeno. Pokud se na tomto místě nachází objekt reprezentující směrovou dlaždici s motivem šipky určující směr (obrázek č. 9), je v momentě stoupnutí číšníka doprostřed této dlaždice změněn parametr směru. To má za následek otočení číšníka daným směrem a číšník pak pokračuje v tomto směru ve vykonávání pohybu, taktéž v tomto směru.

Obrázek 9: Směrové dlaždice

Přibližný počet dnů strávených řešením tohoto zadání: 1

2.3.8 Změna grafiky pozadí

Zadání

Změnit obrázek použitý v pozadí, za finální verzi poskytnutou firemním grafikem. Vycentrovat hrací plochu tak, aby jednotlivé dílky hrací plochy odpovídaly pozici čtverce na pozadí.

Řešení

Obdržel jsem grafický podklad pro pozadí, který je nyní použit v aplikaci. Jedná se o obrázek čokolády, na níž každý dílek čokolády představuje jeden dílek hrací plochy. Aktualizace tohoto

nového pozadí byla bezproblémová, avšak zmenšit pozadí tak, aby každý dílek odpovídal velikostně dílku hrací plochy, už nebylo tak jednoduché. Musel jsem pro tento případ vytvořit konverzní funkci, která měla za úkol měnit velikost obrázku pozadí tak, aby dílky velikostně odpovídaly. Jelikož Quintus [3] pro změnu velikostí grafiky využíval pouze princip založený na škálování podle určitého koeficientu, implementoval jsem do této konverzní funkce také výpočet tohoto koeficientu v závislosti na velikosti jednoho dílku na obrázku a požadované velikosti dílku aplikace.

Přibližný počet dnů strávených řešením tohoto zadání: 1

2.3.9 Změna grafiky objektů palety

Zadání

Zaměnit stávající obrázky představující objekty, za obrázky vytvořené firemním grafikem.

Řešení

Jelikož poskytnuté obrázky nebyly velikostně stejné a jejich formát nebyl vždy čtvercový, tak řešení tohoto zadání nebylo pouze triviálním vyměněním grafických souborů s obrázky. Musel jsem zde podobně jako u změny pozadí vytvořit funkci, která se starala o škálování obrázků na odpovídající rozměr. Zde už jsem byl schopen v algoritmu koeficient škálování počítat dynamicky z výšky a šířky jednotlivého obrázku a tak zajistit, že všechny obrázky budou mít požadovanou velikost.

Přibližný počet dnů strávených řešením tohoto zadání: 1

2.3.10 Animace čišníka

Zadání

Vytvořit animace čišníka z poskytnutých grafických podkladů, které zobrazují jednotlivé fáze pohybu v určitých směrech a situacích.

Řešení

Jelikož Quintus [3] poskytuje podporu animací, vytvořil jsem z dodaných grafických podkladů potřebné definice animací, jenž Quintus [3] vyžaduje. Některé z obrázků použitých pro animace čišníka jsou vyobrazeny na obrázku č. 10.

Animacím tak můžeme nastavovat:

- jednotlivé obrázky pro přehrávání
- rychlost přehrávání
- animaci, jenž má následovat po skončení přehrávání aktuální animace
- funkce, která se má spustit po ukončení přehrávání
- a také možnost zda chceme, aby se animace provedla pouze jednou nebo se opakovala

Pro aplikaci byly potřebné animace pohybu v různých směrech, otáčení a animace konce pohybů, pokud se měl číšník zastavovat. Tyto vytvořené definice pak fungují jako funkce, které

se volají v potřebných místech. Jakmile se dá číšník do pohybu je zavolána animace odpovídající aktuálnímu pohybu, který číšník vykonává.

Při implementaci animací jsem se setkal s problémy, které se týkaly načasování pouštění, přehrávání a zastavování animací tak, ať dávají smysl dohromady s pohybem číšníka. V některých případech jako například při změně směru, jsem musel zastavit pohyb číšníka, přehrávat animaci otáčení a až po skončení animace otáčení číšníka zase rozpohybovat a pouštět animaci pohybu v daném směru.

Obrázek 10: Číšník

Přibližný počet dnů strávených řešením tohoto zadání: 4

2.3.11 Zobrazení zlomku u číšníka a koberce

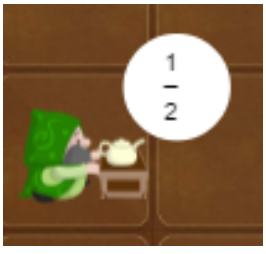
Zadání

Zobrazovat hodnoty zlomků u číšníků a koberců (představující zákazníky) v malé bublince, která se v případě číšníka bude pohybovat spolu s ním.

Řešení

Využil jsem metody *draw()* [6] používané pro vykreslování grafiky objektů v herním jádru Quintus [3] a tuto metodu jsem rozšířil. Nadefinoval jsem v této metodě vykreslování vyplněného kruhu potřebné velikosti v pravé horní pozici od objektu. Dále jsem do tohoto kruhu, představující bublinku, nastavil vypisování aktuálního zlomku, jenž objekt nese. Zlomek je tak zobrazován ve formě čitatele, jmenovatele a krátké linky mezi nimi, jenž představuje lomítko zlomku. Výsledná bublinka obsahující zlomek je zobrazena na obrázku č. 11.

Jelikož se metoda *draw()* [6] provádí nepřetržitě, aby zajistila vykreslování aktuální grafiky, bylo tak zajištěno automatické zobrazování aktuálního zlomku ihned i po jeho změně. Ať už se jedná o změnu uživatelem nebo přepočtem zlomku při nalévání kávy, které je zajišťováno samotnou aplikací.



Obrázek 11: Zobrazování zlomků

Přibližný počet dnů strávených řešením tohoto zadání: 1

2.3.12 Zrychlené přehrávání

Zadání

Přidání ovládacího prvku do palety pro zrychlení aplikace včetně implementace funkcí a logiky.

Řešení

Do palety jsem přidal tlačítko – HTML element s hodnotou zobrazující zda je zapnut normální nebo zrychlený režim simulace roznášení kávy. Vytvořil jsem funkci, která zajišťovala změnu informace o rychlosti přehrávání v aplikaci. Tuto funkci jsem následně přiřadil do události, která zachycuje kliknutí na toto tlačítko. V metodě pohybu a přehrávání animací číšníka jsem zajistil, že při změně této informace je automaticky přizpůsobena rychlost pohybu a přehrávání animací. V aplikaci jsou tedy 2 různé rychlosti přehrávání: normální a zrychlené.

Přibližný počet dnů strávených řešením tohoto zadání: 0,5

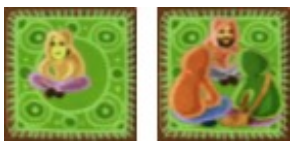
2.3.13 Zobrazení postaviček na koberci

Zadání

Při položení objektu koberce na hrací plochu na něj rozložit postavičky (zákazníky) podle zadaných variant.

Řešení

Pro realizaci tohohle zadání jsem si musel programově vytvořit nový obrázek. Toto jsem provedl vytvořením nového plátna (canvasu) do aplikace. Tomuto plátnu jsem nastavil rozměry objektu koberce a vložil do nejnižší vrstvy obrázek koberce. Pak jsem vytvořil algoritmus pro generování jednotlivých variant zobrazení postaviček podle zadání. Pomocí toho algoritmu se generují postavičky představující zákazníky sedící na koberci a ty jsou vloženy do vytvořeného plátna s kobercem. Takto vytvořený obrázek se předává metodě *draw()* [3], která se stará o vykreslování grafiky objektu, jak jsem už psal výše. Výsledkem je tak mnoho různých kombinací počtů, vzhledů a pozic postaviček sedících na koberci. Dvě z těchto mnoha kombinací jsou vyobrazeny na obrázku č. 12.



Obrázek 12: Koberce s náhodnými postavičkami

Přibližný počet dnů strávených řešením tohoto zadání: 3

2.3.14 Nastavovací formulář

Zadání

Vytvoření vyskakovacího formuláře, který bude určen pro nastavení aplikace. Ve formuláři zobrazovat řádky s objekty, které jdou umísťovat na hrací plochu.

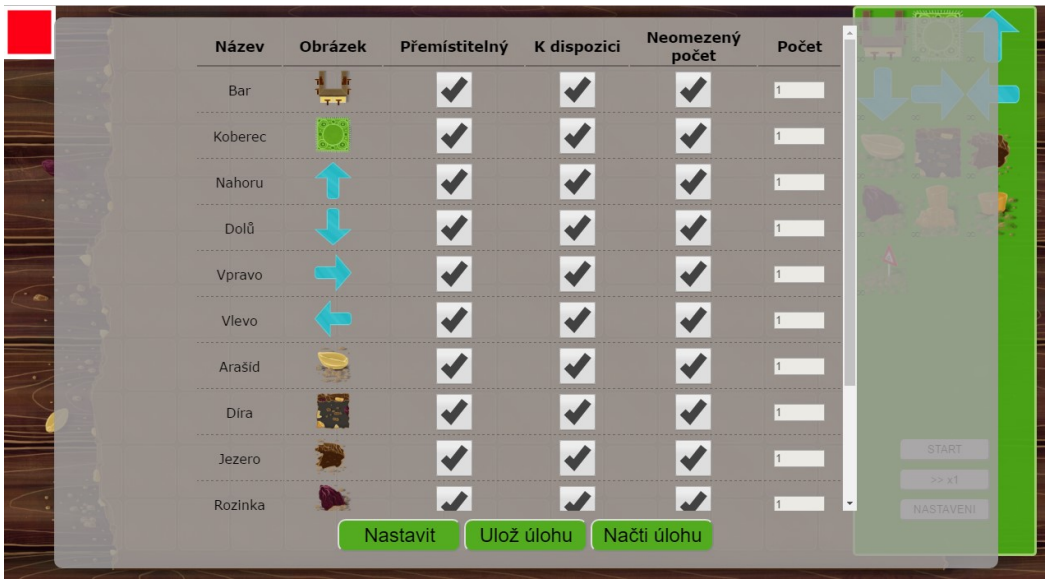
Každý řádek bude obsahovat:

- název objektu
- jeho obrázek
- zaškrtávací tlačítka pro:
  - přemístitelnost objektu
  - informace zda bude objekt k dispozici
  - neomezený počet
- počet použitelnosti objektu v rozmezí 1-99 (pokud není zaškrtnut neomezený počet)

Všechna nastavení se mají aplikovat po stisknutí tlačítka „Nastavit“ ve formuláři, a formulář se má poté schovat.

Řešení

Vytvořil jsem HTML sktrukturu pro formulář obsahující tabulku. Naimplementoval jsem funkce, která do zmíněné tabulky dynamicky vygeneruje obsah pro každý řádek, podle seznamu objektů definovaných výčtem prvků nastaveným v aplikaci. Tento formulář jsem graficky upravil pomocí CSS a nastavil mu i responzivní chování, které reaguje na změnu velikosti okna a zobrazuje obsah tak, aby byl i na menších zařízeních použitelný a čitelný. V tomto formuláři je pak možné nastavovat požadované parametry. Všechny nastavené změny se pak ihned projeví v paletě s objekty po stisknutí tlačítka „Nastavit“. Zobrazení palety s aktuálním nastavení jsem realizoval jejím překreslením podle nastavených parametrů. Finální vzhled nastavovacího formuláře je zobrazen na obrázku č. 13.



Obrázek 13: Formulář pro nastavení aplikace

Přibližný počet dnů strávených řešením tohoto zadání: 3

2.3.15 Ukládání a načítání úlohy

Zadání

Implementovat ukládání a načítání úlohy i s jejím nastavením pomocí specifických tříd, JSON objektů a souborů s touto strukturou.

Řešení

Nejdříve jsem si vytvořil několik tříd, které představovaly strukturu datových kontejnerů pro jednotlivé informace o objektech a nastavení těchto objektů i celé aplikace. Některé instance tříd tak obsahovaly pole instancí tříd jiných. Díky této struktuře a deklaraci všech potřebných parametrů jsem byl poté schopen uložit všechny nezbytné informace o stavu celé aplikace do jednoho objektu a s tím pak dále pracovat.

Pro ukládání a načítání stavu aplikace jsem přidal do formuláře s nastavením HTML tlačítka „Ulož úlohu“ a „Načti úlohu“.

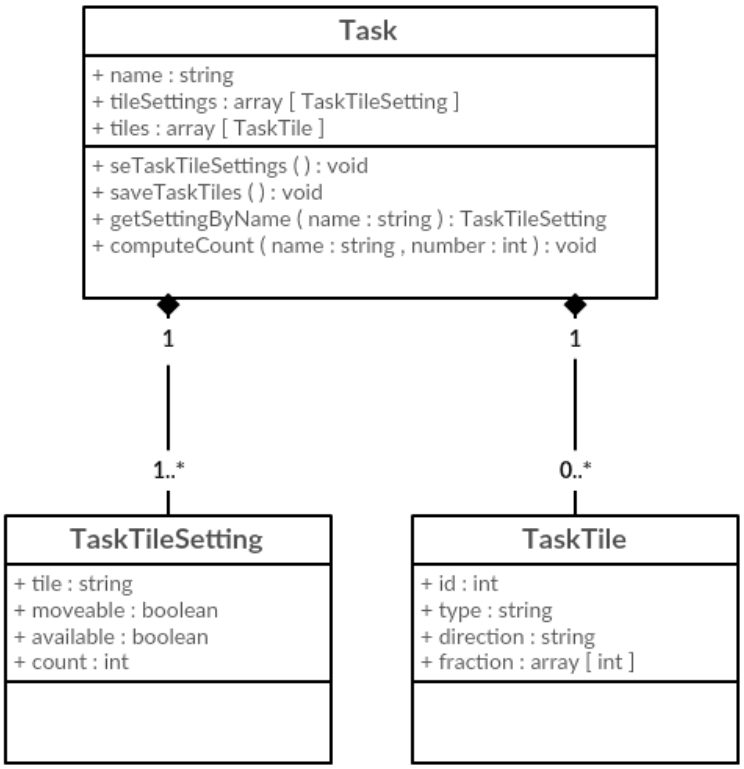
Pro ukládání úlohy, jsem si do připraveného objektu s potřebnou strukturou uložil všechny potřebné informace jako pozice, hodnoty a nastavení objektů a palety. Tento objekt je následně při kliknutí na tlačítko pro uložení úlohy převeden do JSON formátu. Jelikož jsem neměl za úkol implementovat ukládání do souboru na disk nebo posílání JSONu na server, kde by byl dále zpracován a uložen, je tento JSON pouze vypisován do konzole prohlížeče.

Pro načítání uložené úlohy aplikace také využívá podobného principu jako při ukládání.

Pro prvotní inicializaci aplikace je použit JSON soubor uložený fyzicky v adresářové struktuře společně s aplikací. Soubor obsahuje výchozí nastavení aplikace. Toto výchozí nastavení je možné měnit úpravou tohoto souboru popřípadě jeho výměnou. K načtení souboru jsem využil AJAXové funkce. Jelikož tato funkce načítá soubor asynchronně, měl jsem při realizaci načítání ze souboru problém, kdy se mi aplikace a načítání nastavení spustilo dříve, než byl požadovaný soubor načten. Tento problém jsem vyřešil „callback“ funkcí. To znamená, že jsem do AJAXové funkce poslal hlavní funkci aplikace jako parametr. To zapříčinilo, že se aplikace spouští až po úspěšném načtení souboru.

Pro načtení uložené úlohy po tom, co je již provedena prvotní inicializace, slouží jednoduchý formulář zobrazen při kliknutí na tlačítko pro načtení úlohy. Jelikož jsem ani zde neměl zadáno řešit komunikaci se serverem nebo dialogové načítání souboru z počítače, je zatím načtení prováděno vložením textu ve formátu JSON do formuláře a jeho potvrzením. Po potvrzení je uživatel informován o načtení, je zobrazena požadovaná úloha a aplikováno nastavení paleta s objekty.

V obrázku č. 14 je zobrazen diagram tříd, pomocí kterých je realizováno ukládání a načítání úlohy. Třída „Task“ reprezentuje úlohu. Třída „TaskTileSetting“ reprezentuje obecné nastavení pro všechny objekty v paletě, které je možné přesouvat na hrací plochu. Třída „TaskTile“ reprezentuje konkrétní objekty již rozmístěné na hrací ploše.



Obrázek 14: Třídní diagram – ukládání a načítání úlohy

Přibližný počet dnů strávených řešením tohoto zadání: 4

2.3.16 Implementace logiky pro nalévání kávy

Zadání

Implementovat logiku nalévání kávy číšníkem. Jakmile projde číšník přes koberec, odečte se mu ze zlomku odpovídající počet, který koberec vyžaduje a číšník pak dále pokračuje ve své cestě. V případě, že číšník nedonese dostatek kávy, aby uspokojil požadavek stolu, předá všechnu kávu a také pokračuje dále. Jakmile jsou všichni zákazníci na koberci obslouženi (koberec nevyžaduje už žádné množství kávy), tímto kobercem číšník pouze projde.

Řešení

K řešení tohoto zadání mi byla poskytnuta JavaScriptová knihovna třetí strany, která obsahovala implementaci základních matematických operací pro práci se zlomky. Objektu koberec jsem vytvořil metody, jenž se starají o výpočet množství donesené kávy a zajišťují i následné zobrazování provedených změn ve zlomcích. Tyto metody jsou pak vyvolány v moment, kdy číšník dojde doprostřed koberce.

Požadavkem bylo také vizuálně pomocí barvy bublinky se zlomkem odlišit stav koberce. Pokud zákazníkům není doneseno požadované množství kávy, má bublinka koberce se zlomkem červenou barvu. Jakmile je doneseno dostatečné množství a koberec je plně obsloužen, barva bublinky zezelená.

Přibližný počet dnů strávených řešením tohoto zadání: 2

2.3.17 Animace nalévání

Zadání

Zobrazovat animaci nalévání kávy do hrníčků podle zadaných grafických podkladů. Podmínkou bylo, aby zobrazované animace nalévání kávy přibližně odpovídaly vizuálně skutečným hodnotám zlomků.

Řešení

Nad tímto úkolem jsem strávil nejvíce času a byl pro mě zároveň nejtěžším a nejrozsáhlejším zadáním z celé praxe. Maximální počet hrníčků v animacích je 12 a tím jsem byl omezen zobrazovat graficky pouze zlomky do 12/1. Každá fáze nalití jednoho hrníčku pak odpovídá pětinám. Jelikož požadavkem bylo vykreslovat animaci spojením jednotlivých vrstev (pozadí, hrníček, káva), vytvořil jsem si pro každou vrstvu definici animací podle zadaných parametrů.

Aplikace však umí pracovat i se zlomky přesahující 12/1, a proto jsem musel naimplementovat algoritmus, který přepočítával aktuální zlomky tak, aby se animace vizuálně přibližovaly reálným číslům. Jakmile tedy číšník dojde na koberec a má proběhnout nalévání kávy, všichni číšníci se zastaví a začne nalévání. Spustí se funkce pro výpočet zlomků a zároveň začne přehrávání animací. Algoritmus pro přehrávání vypočte rozsah animací, určí od které animace má začít přehrávání a spojením několika vrstev obrázků tak přehraje výslednou animaci nalití kávy (obrázek č. 15 znázorňuje jednu část výsledné animace). V případě, že už byla nějaká káva donesena, animace pokračuje tam, kde ta předchozí skončila. Implementace samotného přehrávání animací podle vypočtených rozsahů je zobrazena ve výpisu zdrojového kódu č. 2.

Jelikož se animace pro jeden hrníček skládá pouze z pěti částí, setkal jsem se s problémem, kdy například nebylo doneseno takové množství kávy, aby pokrylo jednu pětinu animace a při následujícím dolití kávy se animace již nezobrazovaly správně. Na podobné problémy jsem narazil během implementace ještě několikrát, a proto jsem musel algoritmus mnohokrát upravovat.

Obrázek 15: Animace nalévání kávy

Ve výpisu zdrojového kódu č. 2 je zobrazena metoda *step()* [6] objektu, který představuje jednu vrstvu animace nalévání. V této metodě jsou volány další vytvořené funkce pro spouštění animací a jejich zastavování. Implementoval jsem do metody také odpočet času po skončení animace, pro uvedení číšníků zpět do pohybu.

```
step: function (dt) {
    if (Running === true) {
        if (this.p.currentAnim <= this.p.numberOfAnimations) {
            if (this.p.animationInProgress === false) {
                this.p.animationInProgress = true;
                if (this.p.animateDebug) {
                    debug("played animations: " + this.p.currentAnim
                            + " - " + pouringAnimList[this.p.currentAnim]);
                }
                Q("Cisnik", 2).trigger("pauseMove");
                this.play(pouringAnimList[this.p.currentAnim]);
            }
        }
    } else {
        this.pouringWasComplete();
    }
    if (this.p.pouringDone) {
        this.p.waitTime -= dt;
        if (this.p.waitTime < 0) {
            this.p.waitTime = pouringEndWaitTime;
            this.p.pouringDone = false;
            this.pouringWasComplete();
        }
    }
};
```

Zdrojový kód 2: Přehrávání animace jedné vrstvy

Přibližný počet dnů strávených řešením tohoto zadání: 9

2.3.18 Opravy chyb a ladění

Mým posledním úkolem na praxi bylo odstraňování nedostatků, chyb a implementace drobných změn v aplikaci. Všechny tyto verze aplikace už jsem nenahrával pouze na systém SVN, ale také na server, kde aplikace běží v testovacím prostředí a je přístupná z internetu. Pro nahrání na tento server jsem použil komunikaci přes FTP.

Přibližný počet dnů strávených řešením tohoto zadání: 3

Obrázek č. 16 zobrazuje běžící aplikaci se spuštěnou simulací rozlévání kávy. V levém horním rohu je vyobrazen indikátor stavu simulace. V pravé části se nachází paleta s některými ovládacími prvky aplikace a se všemi použitelnými objekty, které je možné vložit na hrací plochu. Na pozadí aplikace je vidět hrací plocha, na které jsou některé objekty z palety rozmístěny a také číšníci nesoucí kávu.

Obrázek 16: Spuštěná aplikace

3 Závěr

V průběhu absolvování odborné praxe ve společnosti EDIacademy s.r.o. jsem vytvořil aplikaci, pomocí které je uživatel schopen připravit početní úlohu pro výuku a procvičování matematických operací týkajících se sčítání a odečítání zlomků. Tyto početní operace jsou realizovány formou hry, ve které číšníci roznáší hostům kávu reprezentovanou zlomky. Početní úloha je vytvořena rozmístěním předmětů z palety objektů na hrací plochu a nastavením hodnot kávy (pokud to daný objekt umožňuje). V nastavení aplikace je možné omezovat počty objektů, které jsou vkládány do hracího pole a také nastavit, zda mohou být přesouvány či nikoliv. Pokud je povoleno přesouvání daného objektu, je tento objekt možno nejen přesouvat po hrací ploše, ale také z plochy odstranit. Vytvořené úlohy s aktuálním nastavením aplikace a všemi hodnotami je možné ukládat a načítat.

Při práci na aplikaci ve firmě jsem uplatnil některé znalosti získané během studia na vysoké škole. Využil jsem především znalosti týkající se objektově orientovaného programování. V praxi jsem tak měl možnost efektivně využít toto programovací paradigma. V průběhu psaní samotného kódu a implementace aplikační logiky jsem poznal výhody OOP, mezi které patří práce s objekty a programování ve funkcích, které si předávají parametry. Jako menší nevýhoda se mi připadala situace, kdy jsem potřeboval globálně uchovat určitou informaci v aplikaci. Toto uchovávání stavu aplikace je dle mého názoru lépe řešeno programováním funkcionálním.

V průběhu praxe jsem se poprvé setkal s uchováváním informací v JSON formátu, které se hojně využívají nejen v oblasti webových aplikací. Dále jsem měl možnost pracovat se systémem Subversion a poznat jeho výhody i nevýhody. Získal jsem mnoho zkušeností při práci s JavaScriptem a díky zkušeného vedení jsem získal i cenné rady jak správně strukturovat samotný kód, aby byl přehlednější, úspornější a lépe čitelný. Podrobně jsem se seznámil s herním jádrem Quintus a objevil jeho silné i slabé stránky.

Z hlediska získávání informací o technologiích potřebných při tvoření aplikace, jsem většinou neměl problém. Jednak z důvodu práce pod zkušeným mentorem, tak i z důvodu toho, že většina technologií je světově rozšířena a využívána. Setkal jsem se však s menším problémem týkajícího se informací o technologii Quintus. Technologie má dostatečnou dokumentaci z hlediska jak ji využívat. Jakmile jsem však potřeboval konkrétnější informace ke specifickým méně známým funkcím a metodám, bylo obtížné je najít nebo byly informace velmi strohé. Tento fakt občas trochu zpomaloval vývoj aplikace.

Díky praxi jsem tak měl možnost proniknout do světa vývoje webových aplikací, ke kterým jsem se během studia příliš výrazně nedostal. Vyzkoušel jsem si tak práci s nejrůznějšími technologiemi z oblasti webů a získal cenné zkušenosti, které budu moci využít i v budoucnu.

4 Zdroje

[1] EDIacademy s.r.o. [online]. [2017-04-20]. Dostupné z: *http://www.ediacademy.cz/*

[2] H-mat, o.p.s. [online]. [2017-04-20]. Dostupné z: *http://www.h-mat.cz/*

[3] Quintus. [online]. [2017-04-20]. Dostupné z: *http://www.html5quintus.com/*

[4] Quintus. Documentation. [online]. [2017-04-08]. Dostupné z: *http://www.html5quintus.com/documentation*

[5] Quintus. API. [online]. [2017-04-08]. Dostupné z: *http://www.html5quintus.com/api/*

[6] Quintus. Working with Sprites. [online]. [2017-04-20]. Dostupné z: *http://www.html5quintus.com/guide/sprites.md*

[7] HTML Introduction. [online]. [2017-04-20]. Dostupné z: *https://www.w3schools.com/html/html_intro.asp*

[8] HTML5 Introduction. [online]. [2017-04-20]. Dostupné z: *https://www.w3schools.com/html/html5_intro.asp*

[9] W3C HTML5. [online]. [2017-04-20]. Dostupné z: *https://www.w3.org/TR/html5/*

[10] CSS Introduction [online]. [2017-04-20]. Dostupné z: *https://www.w3schools.com/css/css_intro.asp*

[11] MDN JavaScript. [online]. [2017-04-08]. Dostupné z: *https://developer.mozilla.org/cs/docs/Web/JavaScript/*

[12] jQuery. [online]. [2017-04-08]. Dostupné z: *https://jquery.com/*

[13] GIMP. [online]. [2017-04-20]. Dostupné z: *https://www.gimp.org/*

[14] jQuery UI. Draggable. [online]. [2017-04-20]. Dostupné z: *https://jqueryui.com/draggable/*

[15] jQuery UI. Droppable. [online]. [2017-04-20]. Dostupné z: *https://jqueryui.com/droppable/*