INFORMATIKA

Úlohy pro výuku JavaScriptu z předmětu Programování 1 (2. díl)

FILIP FRANK – MIROSLAV ZÍKA

Fakulta pedagogická ZČU v Plzni

Představujeme zbývající úlohy používané v rámci předmětu PGM1P na Katedře výpočetní a didaktické techniky na Fakultě pedagogické při Západočeské univerzitě v Plzni. V prvním díle jsme si představili kompletní seznam úloh, ale z důvodu rozsahu jsme nemohli představit všechny úlohy. Zbývající úlohy popisujeme v tomto příspěvku a to, stejně jako v minulém případě, včetně opakovaně pozorovaných chyb. V prvním dílu jsme se zároveň seznámili s požadavky na studenty, očekávanými výstupy a zařazením z pohledu RVP, neboť se jedná o budoucí učitele informatiky.

Představené úlohy jsou používané při výuce studentů bakalářského studia. Z důvodu velmi rozdílných zkušeností studentů s programováním je zapotřebí začít s výukou programování od skutečných základů. Pro příklad rozpětí si můžeme uvést, že mezi studenty se řadí absolventi gymnázií, absolventi elektrotechnických škol, ale také absolventka oboru cukrářství.

Ovocné hrátky s polem

Tato úloha si klade za cíl seznámit studenty s prací s polem. Protože se jedná o první využití array, tak primárním cílem je studentům představit připravené metody, které jazyk JavaScript při manipulaci s poli využívá. Bude se jednat zejména o přidávání prvků, jejich odstraňování, prohazování nebo získávání dat z pole. Zároveň by studenti měli pochopit, proč je pro ně pole výhodnější než třeba string, který jsme používali v úkolu

se smilíky. Zmíněná úloha by totiž ve své podstatě byla pomocí pole lépe řešena.

Jako v jiných úlohách, ani v této se nebudeme zabývat CSS. Studenti si na začátku připraví HTML strukturu podle předlohy a až následně budeme vše „oživovat“. Vše, co budou studenti pro rozvržení HTML částí potřebovat je tag
.

V JavaScriptu studenti nejprve deklarují pole a rovnou jej naplní čtyřmi výchozími hodnotami. Aby nemuseli zbytečně přemýšlet nad slovy vyplňující pole, využíváme k jeho naplnění názvy ovoce. Prvním úkolem, který studenti s polem budou realizovat, je jeho výpis. Zde hraje významnou roli využívaný programovací jazyk (např. jazyk PHP používá *implode(separator, array): string*). JavaScript k tomuto účelu využívá funkci *Join()*, jejímž jediným parametrem je oddělovač jednotlivých hodnot. Vzhledem k tomu, že řešení považujeme za poměrně jednoduché a snadno zjistitelné, studenti funkci vyhledají a pokoušejí se aplikovat zcela sami.

Další částí úlohy je výpis konkrétního prvku podle zadané číselné hodnoty. Zde je nutné si uvědomit, že pole se indexují od 0 a ne od 1. Pokud budeme mít pole „*ovoce = ["Banány","Pomeranče","Jablka","Hrušky"]*“ a uživatel zadá číslo 2, dojde k výpisu slova „Jablka“. Studenti by však měli upravit uživatelský vstup tak, aby výstup odpovídal konvenčnímu způsobu počítání od jedné. Pokud tedy uživatel zadá 2, měli by studenti zařídit, aby byl vypsaný „Pomeranč“.

Dále studenti musí dokázat přidat prvek na konec nebo na začátek pole. Obecně doporučujeme studentům začít přidáváním na poslední pozici. Tato problematika má dvě řešení – buď je možné zjistit délku celého pole pomocí funkce *length()* (vrácená je vždy o jedna větší, než je nejvyšší index) a na tento index umístíme nový prvek. Pole se nám tím rozšíří o nový prvek, který bude umístěn na konci. Druhou možností je v JavaScriptu připravená funkce *Push()*, která rovněž přidává prvek na konec pole. Výběr použité metody je čistě na uvážení studentů, nicméně jsme vypozorovali větší frekvenci funkce *Push()*. Navíc postup, který využívá funkci *length()* funguje díky tomu, že všechna pole v JavaScriptu jsou dynamická. Pro přidání na začátek pole pak studenti mohou použít funkci *unshift()*. Zmíněná funkce posune všechny prvky o jeden index výš a následně na první místo umístí zadaný prvek.

Závěrečným úkolem je možnost mazat z pole konkrétní prvky. Pro smazání na začátku mohou studenti využít funkci *shift()* a ke smazání posledního prvku mohou použít funkci *pop()*. Problém nastává při mazání prvku

na jiných indexech, protože přímo k tomu vytvořená funkce není. Nabízí se využití cyklu *while*, který by kopíroval prvek vždy na vedlejší a tím bychom získali na konci pole dva stejné prvky. Poslední bychom pak smazali pomocí existující funkce. Přesto, že neexistuje předpřipravená funkce, je možné řešit tento problém i jednodušeji a sice funkcí *splice()*. Funkce *splice()* je na webu W3Schools popisována jako funkce, která v poli prvky přidává nebo nahrazuje, přičemž navrací pole smazaných prvků. Právě při správném nastavení parametrů je možné funkci *splice()* použít na smazání jakéhokoli prvku z pole. Stačí nastavit první parametr, pomocí něhož s využitím číselné hodnoty stanovujeme index, kde se budeme pohybovat na pozici mazaného prvku. Druhý parametr pak nastavíme na hodnotu 1. Při tomto nastavení bude smazán jeden prvek na indexu určeného pomocí prvního parametru. Hodnotu nejen smaže, ale odstraní celou buňku z pole, takže v poli nevznikají „díry“. (1)

Obr. 1 Úloha *Ovocné hrátky s polem* (Zdroj: Vlastní)

Kód úlohy je k dispozici na https://codepen.io/filip-frank/pen/mGNwEg.

Pozorované chyby

Nejčastější chyby studentů je problém s uvědoměním, že indexy v poli začínají od nuly. Tento fakt nejprve vůbec neřeší. Např. uživatel zadá číslo 2 a místo druhé položky smaže z pole třetí. Následně mají studenti problém pochopit, zda jedničku přičítají, nebo odčítají. Uvědomění si tohoto standardu obvykle trvá ještě několik dalších úloh.

Druhou velmi častou chybou je, že se studenti pokouší pole vypisovat tím, že jej přiřadí rovnou do HTML prvku. Alternativně se jej zkouší přetypovat na datový typ string. Tento způsob funguje, ale pouze částečně, protože mezi jednotlivými prvky následně chybí jakýkoli oddělovač.

Poslední evidovaná chyba se týká mazání prvku na konkrétním indexu. Často mazání prvku z pole realizují pomocí vložení prázdného textu. Na první pohled se sice hodnota z pole může jevit jako smazaná, pokud však pole vypíšeme s oddělovači, objeví se na místě tímto způsobem odstraně-

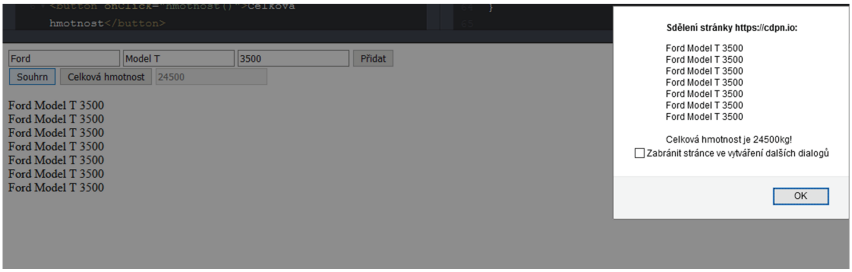
ného prvku oddělovač zdvojeně. Zároveň pak máme problém s adresováním v rámci pole, protože v něm vznikají pomyslné „díry“.

Auta a cykly

Tato úloha si klade za cíl hlouběji seznámit studenty s prací s polem, představit si objekty a prohloubit práci s cykly. Studenti mají za úkol vytvořit pole, do kterého budou následně zadávány značky aut a jejich vlastnosti. Automobily budou vytvořeny jako objekt a jejich vlastnosti budou vlastnosti objektu. Studenti si mimo jiné uvědomí rozdíl mezi běžnou proměnnou, polem a objektem. Názorně uvidí, že by sice bylo možné vytvořit si neomezené množství různých proměnných pro každý vůz a jeho vlastnosti, avšak následná práce s těmito proměnnými by byla velmi těžkopádná.

V průběhu zadávání nových modelů aut se aktualizuje pole s jejich celkovou hmotností a při vkládání nové položky se vypíše na konec výpisu. Zmíněné pole objektů aut by uživatel neměl dokázat měnit. Výsledkem celé práce je vyskakovací okno, které lze vyvolat stisknutím tlačítka souhrn. V něm se vypíší všechny zadané vozy s jejich vlastnosti. Ve spodní části se objeví celková hmotnost všech vozů.

Úplně stejnou úlohu používáme i při zápočtových testech. V testu se pouze úloha změní na *Účtenku* a studenti musí sčítat cenu jednotlivých položek. Na rozdíl od úlohy *Auta a cykly* se po zaplacení vše vynuluje do počátečního stavu, aby „mohl přistoupit další zákazník“. Chyby v účtence jsou totožné s chybami níže.



Obr. 2 Úloha *Auta a cykly* (Zdroj: Vlastní)

Kód úlohy *Účtenka* je k dispozici na https://codepen.io/PGM1Ptest/pen/ZVENVM.

Kód úlohy *Auta a cykly* je k dispozici na https://codepen.io/filip-frank/pen/MzxqPg.

Pozorované chyby

Studenti se vždy pokouší vypisovat pole pomocí funkce *Join()*. Tento postup však není možné použít, protože v poli jsou komplexnější objekty. Funkce umí pracovat pouze s polem, ve kterém jsou hodnoty umožňující převod do datového typu *string*. Při aplikaci na pole se objekty vypíše *[Object object]*. V tomto bodě se často studenti zaseknou a neví, jakým způsobem pole manuálně vypsat. Jen s programováním zkušenější studenti bez problémů využijí cyklus, ostatní musí navést vyučující.

V průběhu používání cyklu FOR je opakující se chybou špatný počet opakování. Studenti často odečítají od délky pole hodnotu 1, čímž si nevypisují poslední hodnotu v poli. Zároveň také zadávají konkrétní hodnotu pro počet průchodů, místo určování počtu průchodů podle měnící se délky pole (vlastnost *length*).

Úvod do canvasu a odrážecí kulička

Úloha *Úvod do canvasu a odrážecí kulička* je úloha, která není úplně konzistentní. Je tvořena jednotlivými ukázkami práce s canvasem, jeho možnostmi a nástroji. Díky tomu si studenti vyzkouší všechny základní nástroje s jejichž použitím mohou vyřešit i komplexní úlohy. Jednotlivé sekce vždy zakomentujeme, abychom mohli vytvořit další část. Pokud by tento postup studentům nevyhovoval, mohou si každou sekci uložit do jiného souboru.

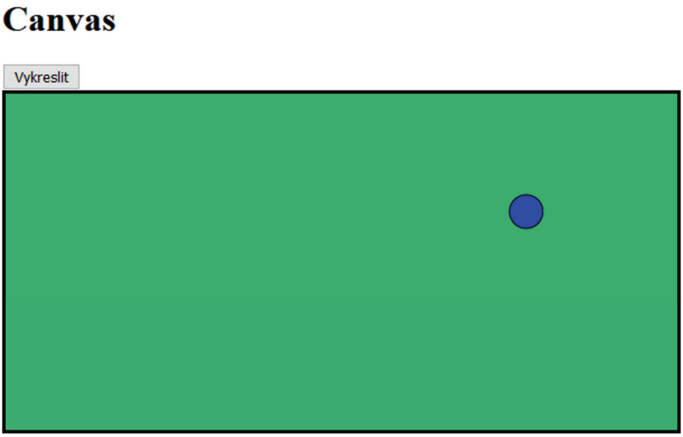
V první části se studenti pokusí vykreslit čtverec pomocí příkazů *lineTo()* a *moveTo()*. Pomocí příkazů nejprve přesunou „Štětec“ do požadované pozice a následně pomocí čtyř čar vykreslí čtverec. Bohužel zároveň zjistí, že takto vytvořený čtverec nemá výplň – je tvořen pouze čárami.

Kvůli nedostatkům prvního pokusu se přesouváme k druhé části, kdy se studenti seznamují s příkazem *rect()*. Tento příkaz vykresluje čtverec nebo obdélník definovaný čtyřmi parametry. První dva parametry udávají pozici levého horního rohu a druhé dva posun. Podle kladných a záporných hodnot tak vykreslíme tvar o námi zvolené velikosti. Následně studenti vytvoří druhý čtverec, který bude o něco níž a bude vyplněn jinou barvou. Zde je potřeba dávat pozor na uzavírání jednotlivých částí. Každý samostatný čtverec by měl být uzavřen mezi příkazy *beginPath()* a *closePath()*. Pokud budou oba čtverce ve stejné sekci, přebarví se nám první na barvu udanou pro druhý čtverec.

Další částí je rozpohybování čtverce. Opět tedy vytvoříme čtverec pomocí příkazu *rect()*, nyní však nebudeme do souřadnice vrcholu zapisovat

konkrétní hodnotu, ale využijeme zde proměnné. Díky tomu můžeme vytvořit zdánlivý pohyb čtverce tak, že vždy vymažeme celé plátno a při každém vykreslení čtverce přičteme k souřadnici osy x zvolenou hodnotu. My jsme zvolili 5 px, které jsou dostatečně velký posun, aby se čtverec hýbal, ale zároveň jde o dostatečně malý skok, aby to vypadalo, jako kdyby se čtverec sunul. Tuto část následně ještě rozvineme o odraz čtverce od konce plátna, jeho souřadnice vždy budou v rozsahu plátna. Zde si je potřeba uvědomit, že pokud máme plátno orámované, kus plátna zakrývá rámeček. Dalším problémem je fakt, že čtverec je nějak veliký. O velikost čtverce tedy musíme posunout hranici odrazu čtverce. Pokud bychom tak neudělali, čtverec by se odrážel svým levým horním okrajem.

Poslední částí je vytvoření kuličky, která se odráží od všech stěn a poletuje po plátně. Nejprve je tedy potřeba nějakým způsobem vykreslit kolečko. Pro jeho vykreslení se využívá příkaz *arc()*, který opět využívá několik parametrů. První dva určují umístění středu kolečka na souřadnicích x a y. Třetí určuje poloměr kolečka, čtvrtý určuje, zda budeme chtít vykreslovat kružnici po směru, nebo proti směru hodinových ručiček a poslední parametr určuje kolik radiánů se z kružnice vykreslí (při zadání *2*Math.PI* se vykreslí kružnice celá). Následně kružnici jen vybarvíme.



Obr. 3 Úloha *Úvod do Canvasu a odrážecí kulička* (Zdroj: Vlastní)

Nyní, když máme připravené kolečko, je potřeba vyřešit jeho pohyb. Místo obou souřadnic na osách zadáme proměnné. Tyto proměnné budeme při každém kroku navyšovat o zvolenou hodnotu. Pokud bude na-

výšení u obou hodnot stejné, bude se kulička pohybovat pod úhlem 45°. Klíčové je správné nastavení okrajů plátna. Vždy když tedy kulička dosáhne stanoveného okraje, je potřeba rozhodnout, zda se jedná o okraj souřadnice x nebo y. Podle toho, o kterou z os se jedná, vynásobíme její posun -1 a tím se kulička odrazí.

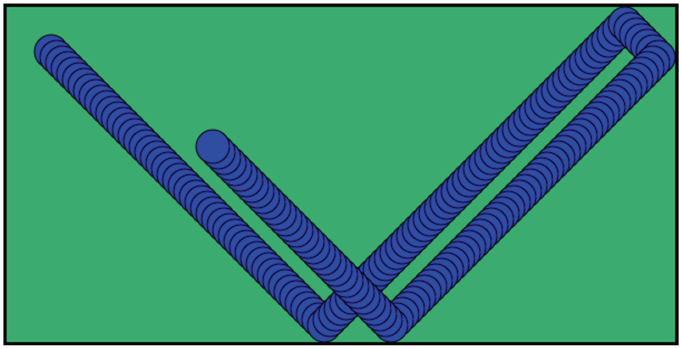
Kód úlohy je k dispozici na https://codepen.io/filip-frank/pen/rNaNerO.

Pozorované chyby

Nejčastěji opakovaná chyba vzniklá při této úloze je, že studenti si připraví celý tvar, ale následně jej nevykreslí. Při dotazu jsme zjistili, že nejde o zapomínání, studenti nevidí důvod pro další vykreslování. Myslí si, že už příkazem *lineTo()* dělají čáru. Z toho důvodu zprvu nevidí důvod použít příkaz *stroke()*. V případě výplně, tedy příkazu *fill()*, tento problém nevzniká.

Při práci s výplní se opakuje jiná chyba a sice, že studenti při rozdílně barevných objektech vybarví oba stejnou barvu. Neuvědomují si, že je potřeba jednotlivé objekty uzavírat. Obvyklý průběh je pak takový, že vybarví první objekt správnou barvou, následně změní barvu a zavolají příkaz pro vybarvení, čímž vybarví druhý objekt. Problémem je, že pokud nejsou objekty odděleny, přebarví se i první.

Klasickou chybou u pohybujících se objektů je zanechání stopy. Studenti zapomínají že to, co namalují do canvasu tam bude zůstávat. V případě pohybujících objektů si neuvědomují, že je třeba vždy canvas smazat a nakreslit objekt s upravenou pozicí.



Obr. 4 Chyba nemazání objektu (Zdroj: Vlastní)

Dále si studenti špatně vypočítávají místo, kde se jim má objekt odrazit. Výsledkem je buď zanořování, výjezd z canvasu, nebo naopak nedosažení úplné hranice, tedy odraz od ničeho.

Poslední opakující se chybou je špatné pozicování. S osou x obvykle studenti nemají problém, protože je pro ně přirozené, že směr doprava je kladný. Přetrvávající problémy se však objevují u osy y, která je pro studenty nepřirozeně rostoucí směrem dolů.

Náhodný graf

Úloha *Náhodný graf* prohlubuje práci s canvasem a ukazuje studentům jedno z praktických využití tohoto HTML prvku. V tomto případě budeme chtít vykreslovat sloupcový graf. Abychom si ověřili, že všechny funkce pracují podle očekávání, budeme pro vykreslování generovat náhodné hodnoty.

Na začátku si studenti připraví HTML strukturu a volitelně si mohou obarvit plátno pomocí CSS. Při další práci už se budeme zabývat pouze JavaScriptem. Prvním problémem je vytvoření pole s náhodnými hodnotami. Abychom předešli v grafu neviditelným sloupcům o nulové hodnotě, omezíme si hodnoty zdola a sice hodnotou 20. Je vhodné omezit hodnoty i shora, abychom viděli vždy vrchol sloupce a nesplynul nám s případným orámováním. V rámci příkladu je zvolena maximální hodnota na 100. Tímto vytvořeným generátorem jsme si nechali vytvořit 14 hodnot, které následně vykreslíme do grafu. Pro kontrolu vygenerovaných si vytvoříme v HTML výpis, kam indexy pole vypíšeme.

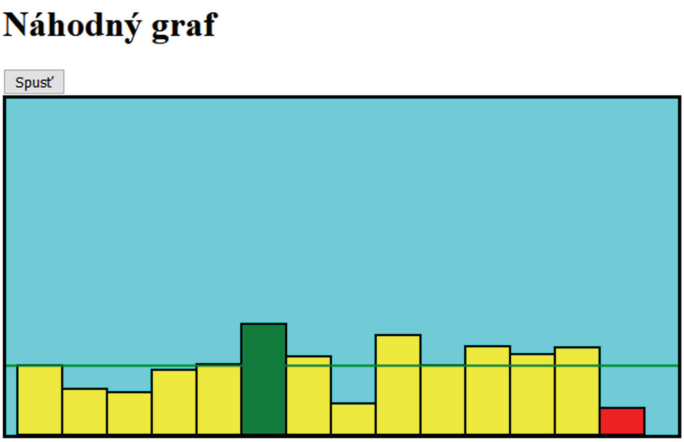
Dále budeme potřebovat vypočítat průměr, maximum a minimum ze všech hodnot. Pro zjištění maximální hodnoty v poli můžeme použít připravenou funkci, která vezme z pole prvek a uloží si jej. Tento uložený prvek porovná vůči vedlejšímu, pokud je vedlejší prvek větší, uloží si ho, v opačném případě zachová původní. Stejným způsobem funguje i funkce pro minimum.

Pro aritmetický průměr si musíme funkci vytvořit sami. Vytvoříme si tedy cyklus s pevným počtem opakování, který projde celé pole a vytvoří sumu hodnot. Výsledek vydělíme délkou pole, čímž získáme průměr. Pokud všechny dosavadní kroky fungují, přesuneme se k vykreslování grafu.

Než začneme pracovat na grafu, je potřeba si uvědomit, že souřadnicový systém počítá obráceně, než budeme chtít vykreslovat graf. Souřadnicový systém funguje tak, že adresa 0,0 je v levém horním rohu a směrem doprava a dolů hodnoty stoupají. Pro vytvoření sloupců odspoda nahoru si

tedy budeme muset uvědomit, že se musíme přesunout na ose y na její maximální hodnotu. V našem případě má plátno rozměr 600×300 px. Souřadnici y tedy nastavíme na 300. Osa x se bude měnit, takže zde zanecháme proměnnou. Šířka sloupce je libovolná, my jsme použili hodnotu 40 px. Nyní zbývá zadat hodnotu z pole. Vzhledem k tomu, že budeme chtít vykreslovat odspoda nahoru, musíme hodnotu vynásobit -1. Díky tomu bude hodnota záporná a od hodnoty 300 se přiblíží vzhůru k 0. Na závěr musíme přičíst k proměnné na ose x hodnotu 40, abychom posunuli příští sloupec o šířku vykresleného sloupce vpravo. Všechny sloupce budeme chtít mít sytě černě orámované a ve výchozím stavu ve žluté barvě.

Pokud při procházení pole program zjistí, že pracuje s maximální hodnotou, vybarví patřičný sloupec zeleně. Při vykreslování minimální hodnoty barva sloupce na červenou. Na závěr vykreslíme zelenou vodorovnou čáru přes celý graf. Zde je nutné si opět uvědomit, že nestačí zadat do osy y hodnotu průměru, ale je potřeba odečíst hodnotu průměru od maximální hodnoty osy y, tedy 300.



Obr. 5 Úloha *Náhodný graf* (Zdroj: Vlastní)

Kód úlohy je k dispozici na https://codepen.io/filip-frank/pen/JjoGbNe.

Pozorované chyby

Studenti mají obvykle problém s datovým typem u proměnné *soucet*. Tato chyba je způsobena tím, že JavaScript si určuje datové typy sám

(většinou je určuje správně). V tomto případě bohužel datový typ správně neurčí. Při výpočtu průměrné hodnoty, kterou získáme dělení součtu počtem prvků v poli, dojde k vyvolání chyby *prumer is NaN*. Proto je nutné při deklarování proměnné do součtu na příklad rovnou uložit 0. Zároveň si tím ošetřujeme, že při opětovném spuštění nebude v součtu předchozí hodnota.

Další opakovanou chybou je obarvení žlutých sloupců zeleně nebo červeně. Tato situace nastane ve chvíli, kdy je špatně uzavřeno kreslení maximálních a minimálních hodnot. Chyba se projevuje buď u všech následujících sloupců, nebo minimálně u sousedního. Podle toho, jakým způsobem je chyba vytvořena.

Někteří studenti se potýkají s problémem, kdy se jim obarví orámování všech sloupců zeleně. Tento problém se objevuje, pokud student nejprve vykresluje průměr, špatně čáru uzavře a pro orámování sloupců neurčí barvu, ale spoléhá na defaultní černou.

Často se objevuje chyba, kdy jsou sloupce malé, takřka neviditelné. Vytvoří prakticky jen rámeček, který se objevuje nad spodním orámování plátna. Tento problém je způsoben špatnou prací se souřadnicovým systémem. Sloupce jsou sice správně velké, ale vedou dolů mimo plátno. S touto chybou se pojí i podobná, kdy je vše správně, jen vedou sloupce z horního okraje plátna dolů. Maximum i minimum bývá v těchto případech správně, pouze jsou v grafu shora dolů.

Opakovanou chybou je vykreslení všech sloupců na sobě. Tato chyba v některých případech vypadá, že se vykreslilo pouze maximum. Je to způsobeno tím, že studenti zapomínají posunout souřadnice na ose x o šířku sloupce. Je to způsobeno buď tím, že nemění proměnnou nebo místo proměnné použijí hodnotu. U studentů využívající proměnnou jde obvykle jen o zapomenutí. U studentů zapisující hodnotu přímo jde o algoritmickou chybu, kdy si nejsou plně vědomi způsobu fungování.

Poslední opakovanou chybou je špatné umístění čáry průměru a její nesprávná barva. Nesprávná barva je obvykle způsobena opomenutím změny barvy z defaultní na zelenou. Špatné umístění je způsobeno tím, že student neodčítá hodnotu průměru od 300 – čára průměru je tedy vysoko nad grafem.

V této úloze se pak objevuje studenty domnělá chyba. Při generování náhodných čísel se někdy může stát, že se vygeneruje několik stejných čísel a zrovna jde o maxima nebo minima. V takovém případě, pokud je vše správně naprogramováno, se nám objeví několik zelených sloupců nebo

několik červených sloupců. Jedná se o normální běh programu, protože vše jsou to maxima a minima. Studenti však často toto chování považují za špatné vyhodnocení a tráví poměrně dlouhý čas nad opravou. Situace obvykle končí dotazem na vyučujícího.

Závěr

Představená sada příkladů je využita pro výuku programování v jazyce JavaScript na Katedře výpočetní a didaktické techniky. Vzhledem k velmi různorodému předchozímu vzdělání studentů jsou příklady koncipovány tak, aby se při jejich plnění seznámili s nejčastějšími chybami spjatými nejen se zvoleným jazykem, ale také programátorskými chybami způsobené nevhodným postupem (rozdílné datové typy, důležitost správné struktury programu, znovu využitelnost kódu aj.). Díky dodržení principu návaznosti a rostoucí úrovně komplikovanosti dochází k postupnému objevování různorodých cest, které lze využít pro splnění požadovaného cvičení. V rámci předmětu PGM1P je studentům hodnoceno pouze, zda daný příklad funguje, nebo nefunguje, a to i u zkoušky. S ohledem na velmi různorodou vstupní úroveň neřešíme v rámci PGM1P efektivitu kódu, nebo best practices.

Na závěr je nutno podotknout, že předmět PGM1P je zaměřen pouze na základní prvky programovacího jazyka JavaScript a jejich využití v rámci strukturovaného programování. Na tento předmět je navazováno předmětem PGM2B, v němž se studenti seznámí se základy objektově orientovaného programování pomocí obdobně koncipovaných příkladů.

Literatura

[1] Programování 1 pro vzdělávání. Západočeská univerzita, Plzeň.

[2] Rámcový vzdělávací program pro základní vzdělávání. Dostupné z: https://www.edu.cz/wp-content/uploads/2021/07/RVP-ZV-2021.pdf

[3] *Frank, F., Zíka, M.:* Úlohy pro výuku JavaScriptu z předmětu Programování 1. Matematika–fyzika–informatika, roč. 31 (2022), č. 4, s. 303–314.