Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Framework na tvorbu jednoduchých edukačných webových aplikácií pre žiakov 1. stupňa ZŠ

Bakalárska práca

2022
Daniel Hutňan

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Framework na tvorbu jednoduchých edukačných webových aplikácií pre žiakov 1. stupňa ZŠ

Bakalárska práca

Študijný program:	Aplikovaná Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra didaktiky matematiky, fyziky a informatiky
Školiteľ:	PaedDr. Daniela Bezáková, PhD.

Bratislava, 2022
Daniel Hutňan

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Daniel Hutňan
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Framework na tvorbu jednoduchých edukačných webových aplikácií pre žiakov 1. stupňa ZŠ
Framework for creating simple web applications for primary education.

Anotácia: Študent zanalyzuje niekoľko existujúcich edukačných webových aplikácií pre žiakov 1. stupňa, prípadne úlohy v pracovných listoch pre túto vekovú kategóriu – napr. aké typy objektov sú súčasťou zadania, akým spôsobom žiaci zadávajú odpoveď, akým spôsobom sa kontroluje odpoveď a pod. Na základe analýzy navrhne a implementuje modul, resp. skupinu modulov, v ktorých budú definované základné triedy a funkcie, ktoré zjednodušia tvorbu takýchto aplikácií hier tak, aby ich vedel vytvoriť aj programátor začiatočník. Funkčnosť modulu predvedie vytvorením niekoľkých ukážkových aplikácií. Študent zvolí na tvorbu frameworku vhodné nástroje a technológie tak, aby výsledné hry boli buď responzívne webové aplikácie alebo mobilné aplikácie.

Cieľ: Cieľom práce je navrhnúť a implementovať framework na tvorbu jednoduchých edukačných webových aplikácií pre žiakov 1. stupňa a demonštrovať jeho použitie vytvorením niekoľkých ukážkových aplikácií.

Literatúra:
Časopis Infovekáčik rubrika Vyskúšajme sa (http://infovekacik.edu.fmph.uniba.sk/)
Bohunická I. 2018. Pythonovský framework pre vytváranie hier. Univerzita Komenského - Fakulta matematiky, fyziky a informatiky, Bratislava, 2018

Kľúčové slová: framework, webové aplikácie, primárne vzdelávanie

Vedúci:	PaedDr. Daniela Bezáková, PhD.
Katedra:	FMFI.KDMFI - Katedra didaktiky matematiky, fyziky a informatiky
Vedúci katedry:	prof. RNDr. Ivan Kalaš, PhD.
Dátum zadania:	30.09.2021

Dátum schválenia: 06.10.2021

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Ďakujem PaedDr. Daniele Bezákovej, PhD. za vedenie práce, odborné rady a pravidelné konzultácie. Ďalej by som chcel poďakovať testerom knižnice za trefné pripomienky k funkcionalite a v poslednom rade ďakujem svojim rodičom, ktorí mi počas celého štúdia poskytovali financie a morálnu podporu.

Čestné vyhlásenie

Čestne vyhlasujem, že som bakalársku prácu Framework na tvorbu jednoduchých edukačných webových aplikácií pre žiakov 1. stupňa ZŠ vypracoval samostatne s použitím uvedenej literatúry a zdrojov dostupných na internete.

V Bratislave dňa 25.05.2022 .....

Daniel Hutňan

Abstrakt

HUTŇAN, Daniel: Framework na tvorbu jednoduchých webových edukačných aplikácií pre žiakov 1. stupňa ZŠ. [Bakalárska práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Katedra didaktiky matematiky, fyziky a informatiky. – Školiteľ: PaedDr. Daniela Bezáková, PhD.: FMFI UK, 2022, 52 strán

Pri tvorbe jednoduchých edukačných aplikácií často narážame na opakujúce sa postupy. Cieľom tejto práce je vytvoriť knižnicu pre javascript, ktorá zjednodušuje proces tvorby jednoduchých edukačných aplikácií určených pre žiakov 1. stupňa. Inšpiráciu pre funkcionalitu knižnice čerpáme z existujúcich slovenských edukačných materiálov: pracovných zošitov či softvérových aplikácií. Pri návrhu a implementácii kladieme dôraz na jednoduchosť používania knižnice tak, aby sa pomocou nej stále dali vytvárať zložitejšie aplikácie. V rámci práce vytvárame pomocou knižnice súbor ukážkových aplikácií a testujeme jej funkcionalitu. Na konci rozoberáme možné rozšírenia funkcionality knižnice.

Kľúčové slová: javascript, canvas, framework, edukačné aplikácie

Abstract

HUTŇAN, Daniel: Framework for creating simple web applications for primary education. [Bachelor thesis]. Comenius University in Bratislava. Faculty of Mathematics, Physics and Informatics; Department of Didactics in Mathematics, Physics and Informatics. - Supervisor: PaedDr. Daniela Bezáková, PhD.: FMFI UK, 2022, 52 pages

During the production of simple educational applications, we often come across repetitive procedures. The aim of this work is to create a library for javascript, which simplifies the process of creating simple educational applications designed for pupils in primary education. The inspiration for the library's functionality is drawn from existing Slovak educational materials: workbooks or software applications. In the designing and implementing stage, we emphasize the ease of use of the library so that it can still be used to create more complex applications. As part of the work, we create a set of sample applications and test the functionality of the library. At the end, we discuss possible enhancements to the library's functionality.

Keywords: javascript, canvas, framework, educational applications

Obsah

Zoznam obrázkov

Zoznam ukážok kódu

Úvod

Úlohou žiaka na prvom stupni ZŠ je zoznámiť sa so základnými princípmi pri výučbe. Jedná sa hlavne o praktické rozvíjanie logického myslenia, ale aj o uvedenie do konceptov ako sú písmená, slabiky či čísla. Efektívny spôsob, ako sa to dá dosiahnuť, je cez hľadanie asociácií alebo opakujúcich sa vzorov v učive. V prostredí školy sa žiaci učia s pomocou učiteľov a ich výkladov, ale aj za pomoci materiálov ako sú učebnice či pracovné zošity.

Potenciál využitia papierových pomôcok však má svoje medze, hlavne čo sa týka ich interaktivity. Dá sa na nich kresliť či písať, ale možnosti tvorby úloh pre toto médium sú značne limitované.

Tu prichádza do popredia možnosť využiť interaktívne edukačné aplikácie. Okrem toho, že sa žiak učí hravým spôsobom, tvorca učebných materiálov má zároveň širšiu škálu možností pri tvorbe úloh.

Kľúčovým elementom pri tvorbe takýchto aplikácií je doba vývoja. Keďže úlohy sa snažia naučiť jednoduché koncepty a často sa v nich vyskytujú (zámerne) repetetívne princípy, nemá zmysel, aby každá aplikácia bola reprezentovaná stovkami riadkov kódu. Preto je na mieste, aby tvorca týchto úloh mohol stavať na už vytvorených a zorganizovaných prvkoch, ktoré poskladá dokopy podľa svojho uváženia.

Cieľom tejto práce je navrhnúť a implementovať knižnicu pre zjednodušenie tvorby jednoduchých edukačných aplikácií a vytvoriť niekoľko ukážkových aplikácii využitím tejto knižnice. Využitie tejto knižnice by malo byť jednoduché a intuitívne, pričom by sa mal zachovať rozsah možností pri tvorbe aplikácií.

V prvej kapitole analyzujeme pracovné zošity určené pre žiakov 1. stupňa ZŠ a už vytvorené edukačné aplikácie. Taktiež v nej rozoberáme, čo sa dá využiť z už existujúcich bakalárskych prác a aj to, na akých technológiách budeme stavať.

V druhej kapitole popisujeme návrh knižnice, navrhujeme hierarchiu objektov knižnice a jej základnú funkcionalitu. Podľa vytvorenej hierarchie navrhujeme konkrétne triedy a ich funkcionalitu.

V tretej kapitole sa venujeme implementácii knižnice. Rozoberáme, ako jednotlivé triedy zjednodušujú praktické problémy pri tvorbe aplikácií, ako sú triedy medzi sebou

prepojené a ako medzi sebou komunikujú. Na záver kapitoly o implementácii popisujeme proces inštalácie.

V štvrtej kapitole predstavujeme niekoľko jednoduchých ukážkových aplikácií, ktoré sme pomocou našej knižnice vytvorili. Kód týchto aplikácií môže neskôr využiť užívateľ našej knižnice ako odrazový bod pri tvorbe novej aplikácie.

Piatu kapitolu venujeme výsledkom testovania knižnice. Vysvetlíme zmeny v knižnici, ktoré sme na základe pripomienok testerov spravili.

Rozsah našej knižnice si so sebou nesie aj svoje limitácie v tom, čo sa pomocou nej dá implementovať. Možným rozšíreniam knižnice venujeme samostatnú kapitolu.

Kapitola 1

Teoretické východiská

V tejto kapitole predstavíme východiská našej práce, zameriame sa na úlohy z pracovných zošitov, s ktorými sa stretávajú žiaci na 1. stupni ZŠ a analyzujeme už existujúce edukačné aplikácie. Na základe týchto rozborov sa rozhodneme, aké technológie budeme pri implementácii knižnice využívať.

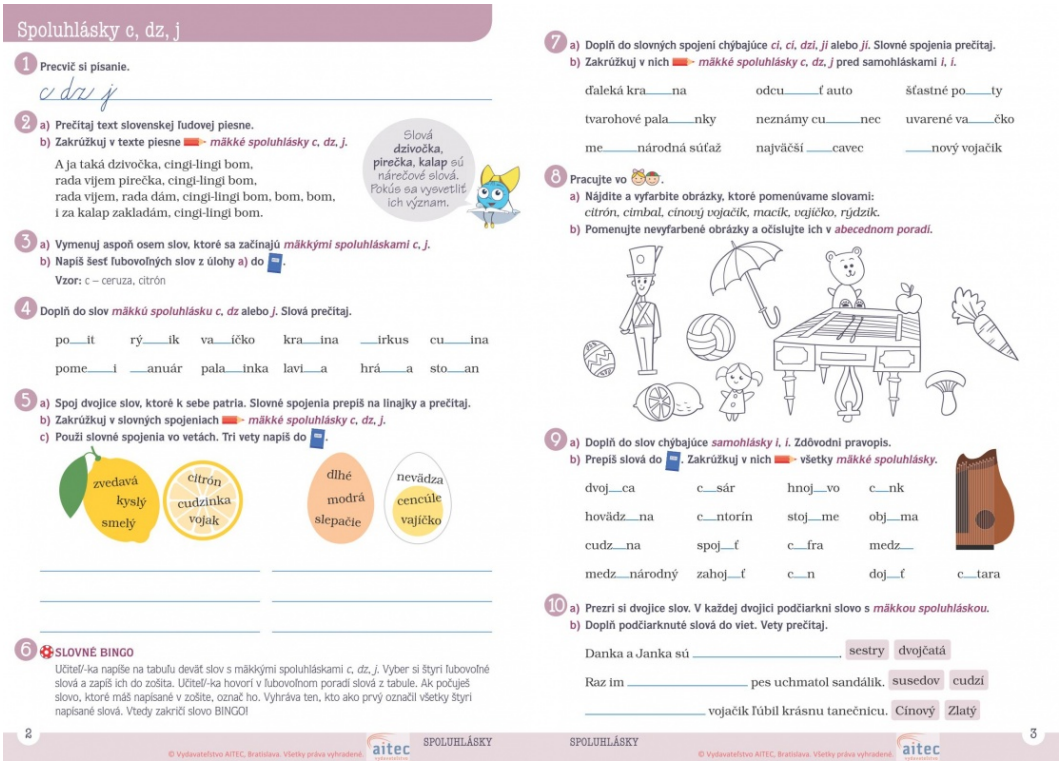
1.1 Pracovné zošity

Pracovné zošity sú na 1. stupni štúdia zatiaľ najpoužívanejšími učebnými materiálmi. V tejto časti kapitoly zaradíme konkrétne úlohy zo zošitov do všeobecných skupín.

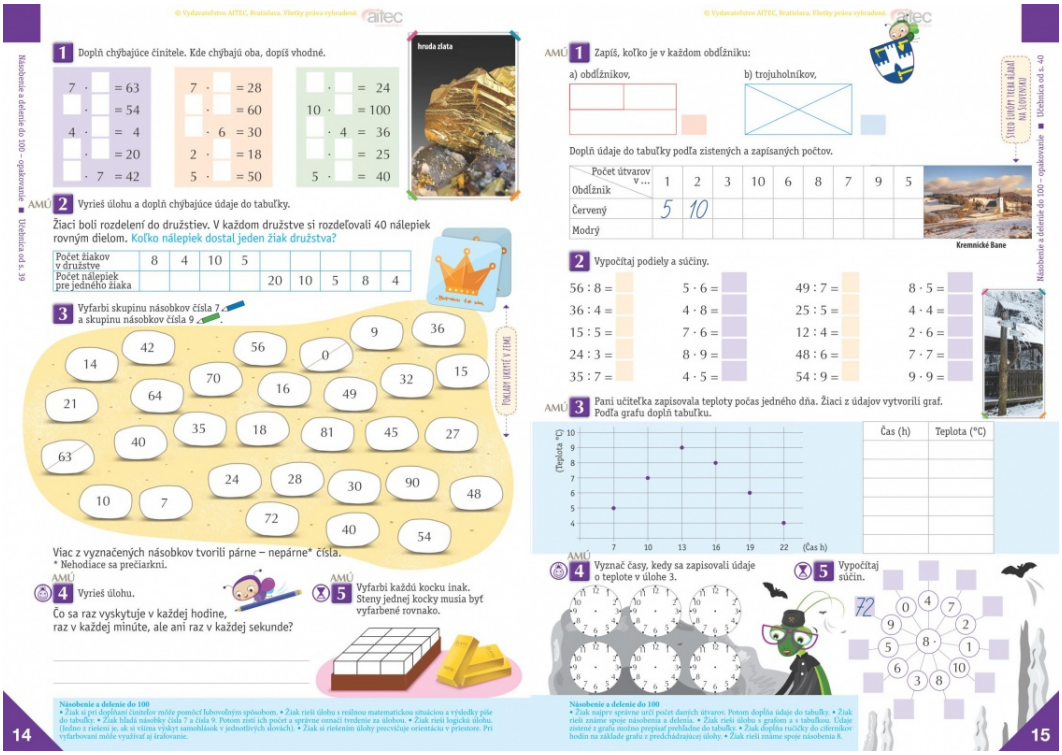
Na obrázku 1.1 vidíme jednu dvojstránku z pracovného zošita na slovenčinu [1]. Úlohy na nej môžme klasifikovať do niekoľkých základných skupín:

- **Doplň správne:** V slove chýba písmeno, treba ho doplniť tak, aby bolo slovo gramaticky správne.
- **Vyber správne:** Žiak má k dispozícii viac slov, má vybrať to, ktoré spĺňa podmienky zo zadania.
- **Zakrúžkuj hlásku:** Žiak si prezrie slová a zakrúžkuje v nich hlásky podľa zadania (napríklad spoluhlásky c, dz, j).
- **Vytvor dvojice:** K dispozícii sú dve skupiny slov. Žiak má za úlohu spojiť tie, ktoré k sebe patria.

Okrem vyššie spomenutých typov úloh sú v zošite na obrázku 1.1 aj úlohy na precvičenie písania alebo kreslenia.



Obr. 1.1: Ukážka pracovného zošita. Slovenský jazyk pre druhý ročník ZŠ.



Obr. 1.2: Ukážka pracovného zošita. Matematika pre štvrtý ročník ZŠ.

Na obrázku 1.2 vidíme jednu dvojstránku z pracovného zošita na matematiku [2]. Opäť môžme úlohy klasifikovať do základných skupín:

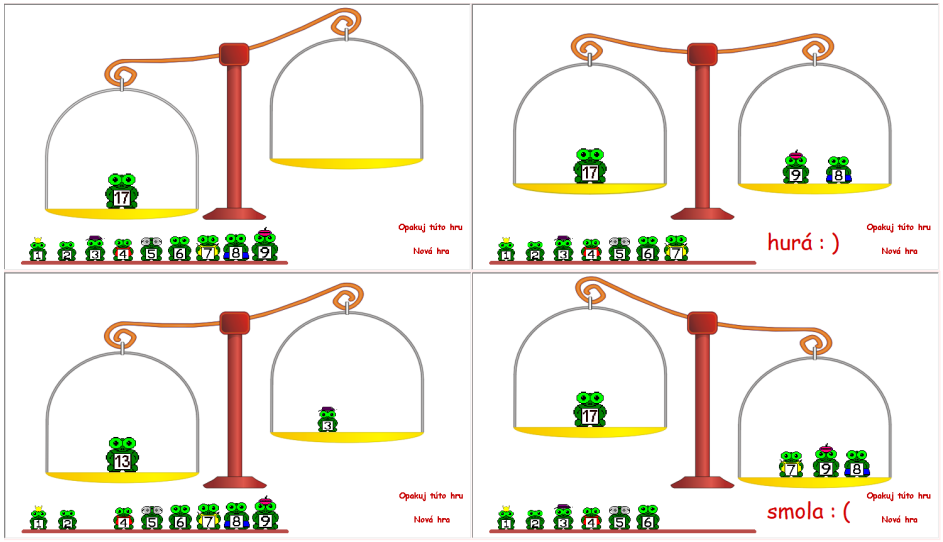
- **Doplň správne:** V príklade chýba číslo. Žiak má doplniť číslo tak, aby dostal správny výsledok.
- **Zaraď do skupín:** Žiak má roztriediť čísla do skupín, napríklad vyfarbiť čísla deliteľné číslom 7 a 9 podľa zadania.
- **Čítaj z obrázka:** Na obrázku je graf. Žiak má z neho vyčítať dáta podľa zadania.
- **Vypočítaj:** Rôzne typy príkladov, žiak má zistiť a zapísať výsledok.

Z tejto analýzy môžme zhodnotiť, že v pracovných zošitoch sa vyskytujú úlohy podobného typu, aj keď sa jedná o rôzne predmety a ročníky. Doplnenie, označenie, spojenie, zaradenie sú úkony, ktoré sa v interaktívnych aplikáciách môžu riešiť klikaním a ťahaním myšou či písaním na klávesnici.

Kým práca s papierovou učebnicou rozvíja schopnosť držať ceruzku, práca na počítači (či inom zariadení) rozvíja schopnosť písať na klávesnici a hýbať myšou.

1.2 Edukačné aplikácie

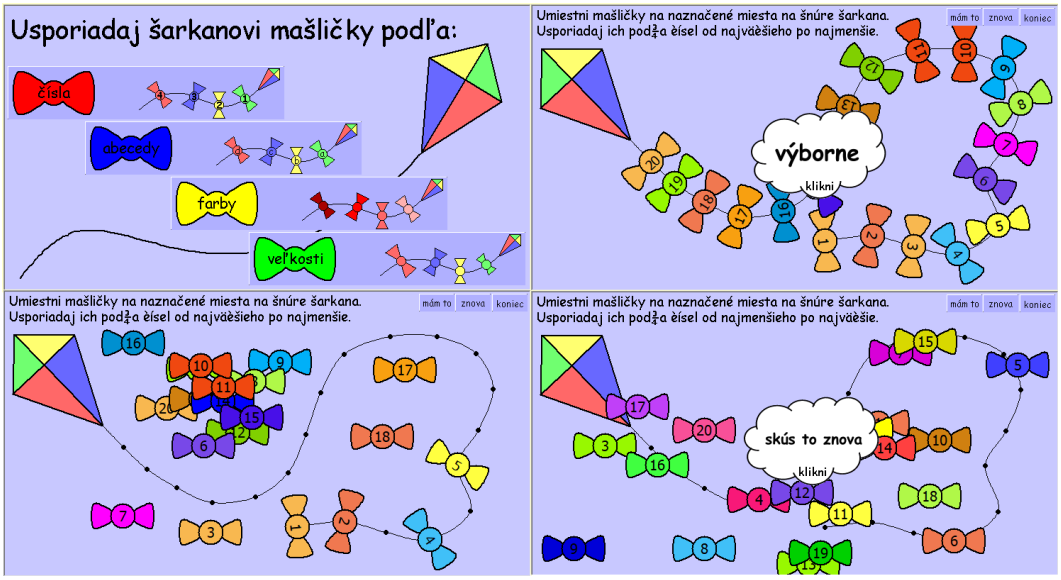
Vytvárame knižnicu na zjednodušenie tvorby edukačných aplikácií, preto treba analyzovať, ako takéto aplikácie fungujú. Jedná sa o logické hry s jednoduchým zadaním. Často sú natoľko intuitívne, že na to, aby hráč (žiak) pochopil zadanie, nepotrebuje žiadne vysvetlenie. Analyzované aplikácie sme čerpali zo zdrojov [3] a [4].



Obr. 1.3: Ukážka aplikácie. Hráč presúva žabky na váhu tak, aby bola vyvážená.

Na obrázku 1.3 je hra, kde hráč presúva objekty po obrazovke. Rozdeľme si hru na komponenty:

- **Tlačidlá:** Reagujú na myš a prevedú nejakú funkciu.
- **Presúvateľné obrázky** (žabky na spodnej lište): Pomocou myši sa dajú chytať a presúvať. Pri pustení obrázka voľne v priestore sa vráti na pôvodné miesto. Pri presunutí obrázka na pravú stranu váh sa váhy naklonia podľa momentálneho zaťaženia.
- **Váhy:** Sú naklonené podla aktuálneho zaťaženia. Reagujú na presunutie objektu na konkrétne miesto. Keď je na oboch stranách rovnaké zaťaženie alebo sa váha príliš preváži, hra končí.
- **Správa o výhre/prehre:** Keď je na pravej strane rovnaké alebo väčšie zaťaženie, hra končí a zobrazí sa text, ktorého obsah závisí od toho, či sú váhy vyvážené alebo nie.



Obr. 1.4: Ukážka aplikácie. Hráč ukladá mašle na šarkana tak, aby boli v správnom poradí.

Na obrázku 1.4 je ďalšia interaktívna aplikácia. Aj keď sa jedná o iné zadanie, technická stránka hry je skoro rovnaká ako tá predošlá.

Opäť sa tam vyskytujú **tlačidlá** s nejakými funkciami, **presúvateľné objekty** - mašle, ktoré treba presunúť na nejaké miesto (bodky na chvoste šarkana) a nejaký **text** - tentokrát so zadaním úlohy.

V aplikácii pribudli nasledovné súčasti:

- **Menu:** Strana, ktorá sa ukáže, keď sa načíta hra, alebo sa k nej dá vrátiť pomocou tlačidla „koniec“.
- **Správa o výhre/nesprávnom riešení:** Zobrazí sa ako obláčik, ktorý „priletí“ z okraja obrazovky (animácia). Reaguje na klik myšou - zmizne.

Obr. 1.5: Ukážka aplikácie. Hráč má spojiť kocky s obrázkami so stavbami z nich.

Na obrázku 1.5 je interaktívna aplikácia zo stránky [4]. Nevyužíva zastaralý plugin, ale vlastnosti HTML5 (tým sa budeme venovať v časti 1.4).

Aj keď vizuálna stránka aplikácie je diametrálne odlišná od predchádzajúcich, princípy ostávajú podobné. Vyskytujú sa tam **tlačidlá**, opäť funkcie na opakovanie hry a kontrolu správnosti, ale je tam aj tlačidlo s kľúčom k riešeniu, tlačidlá, ktoré hru presunú na inú úroveň (v tomto prípade viac kociek, ktoré treba počítať) a tlačidlo, ktoré prečíta text zo zadania. Hra obsahuje aj **obrázky**, ktoré sa ale nedajú presúvať a **text** v hlavičke, aj ako zadanie. Obrázok 1.5 je rozdelený na dve časti: ľavá časť ukazuje začiatok hry, pravá správne riešenie. V ľavom hornom rohu oboch častí je objekt znázorňujúci **stav hry** - v pravej časti je do polovice vyplnený na žlto, čo znamená, že jeden z dvoch levelov je dokončený. Okrem spomenutých častí tam pribudla nasledujúca funkcionalita:

- **Spájanie objektov čiarou:** Hráč klikne myšou do poľa (na obrázku 1.5 ako kruh) a ťahá. Za myšou sa ťahá čiara. Keď myš vstúpi do poľa na druhej strane a hráč ju pustí, polia sú prepojené a aplikácia podľa toho reaguje.
- **Animovaný obrázok** (GIF): Chlapec (v pravej časti herného poľa) sa hýbe a reaguje na akcie hráča (napríklad spájanie, výhra, prehra).
- **Zvuk:** Klik na tlačidlo prehrania zadania prehrá zvukovú stopu. Keď hráč spája objekty, spustí sa zvuk ako reakcia na to. Hra vydáva zvuky aj na pozadí, keď sa nič nedeje.

1.3 Existujúca bakalárska práca

V bakalárskej práci z roku 2018 [5] sa nachádza niekoľko princípov, ktoré môžme využiť aj pri tvorbe našej knižnice:

- Požiadavky
  - Jednoduchosť inštalácie a používania knižnice.
  - Možnosť využiť zložitejšie prostriedky - kontrola nad objektami pri tvorbe aplikácie.
  - Prístup k zmysluplným chybovým správam.
  - Zostava ukážkových hier vytvorených v knižnici.
- Herný objekt Sprite
  - Objekt, ktorý sa vykresľuje na ploche vo forme obrázku, animovaného obrázku alebo tvaru.
  - Dá sa ním manipulovať, aj mu priradiť nejaké funkcie na vykonanie.
- Funkcionalita
  - Reakcia na vstupy hráča: klávesnica, ťahanie myšou.
  - Uchovávanie stavu herných objektov
  - Chybové hlásenia (výnimky)
  - Preddefinované parametre
    - Programátor má možnosť priradiť objektom nastavenia ako natočenie, rýchlosť a smer pohybu. Zmysluplné základné nastavenia zrýchlia čas vývoja a zmenšia rozsah kódu aplikácie.
  - Grid
    - Okrem možnosti umiestňovania objektov podľa súradníc pixelov je pri niektorých typoch hier vhodné pracovať so štvorcovou sieťou. To zjednoduší proces umiestňovania objektov, aj logiku pri ich pohybe.

1.4 Technológie

Knižnicu budeme písať v jazyku JavaScript. Vývojár, ktorý ju bude používať, potrebuje jednoduchý spôsob, ako by ju mohol inštalovať a implementovať do svojej aplikácie. Pre zabezpečenie tejto funkcionality využijeme runtime environment (prostredie behu programu) **Node.js**.

1.4.1 Node.js

Node.js [6] je multiplatformový runtime environment, ktorý presúva exekúciu kódu mimo prehliadača. Tento systém so sebou nesie nástroj **npm** (node package manager), pomocou ktorého sa dajú jednoducho inštalovať balíčky (moduly) z internetu. Tieto vlastnosti ho robia ideálnym nástrojom pre tvorbu aplikácií tak, aby časti boli jednoducho inštalovateľné nezávisle od operačného systému. Nástroj npm tiež sprístupňuje viac ako 350000 balíčkov, na ktorých môže vývojár ďalej stavať. Cez Node.js môže potom vývojár zabezpečovať napríklad celý backend server vrátane databázy. Pre potreby našej knižnice však budú stačiť tieto:

- Express [7]
  - Framework pre webové aplikácie na Node.js.
  - Stará sa o vytvorenie a spravovanie webovej aplikácie, ktorá beží v prehliadači.
  - Zjednodušuje budovanie štruktúry organizácie aplikácií na webe.
- JSDoc [8]
  - Generátor dokumentácie pre JavaScript.
  - Analyzuje kód a automaticky vytvorí čitateľnú dokumentáciu vo forme HTML súborov.

1.4.2 Zobrazovanie aplikácie

Hry, alebo edukačné aplikácie, ktorých tvorbu bude naša knižnica zastrešovať, musia byť na stránke nejako zobrazené. Je niekoľko spôsobov, ako sa to dá dosiahnuť. Zameriame sa na využitie elementov HTML5 a kreslenie na plátno, rozoberieme si ich výhody a nevýhody a vyberieme, ktorý z nich budeme pri implementácii využívať.

- HTML5 elementy
  - Jedná sa o využitie elementov ako `<div>`, `<img>`, `<table>` a ďalších, cez ktoré sa zobrazuje aplikácia. K týmto elementom môže byť naviazaný aj CSS štýl, čím sa dosahuje prepracovanejšia grafická stránka.

  - **Výhody**
    * Možnosť využitia elementov ako `<input>` pre vstup od používateľa.
    * Pekné zobrazovanie prvkov pomocou CSS štýlov.
    * Prehliadače sú optimalizované na využitie týchto elementov - rýchlejší chod aplikácie.
    * Vysoká prístupnosť pre ľudí so zdravotnými obmedzeniami.
  - **Nevýhody**
    * Ak sa vývojár chce odkloniť od predpripravených typov hier, musí pracovať v troch jazykoch (HTML, CSS, JavaScript).
    * Výsledné aplikácie majú zložitú štruktúru (skompilovaný HTML kód aplikácie na obrázku 1.5 má skoro 12000 riadkov).

- Kreslenie na plátno (HTML element `<canvas>`)
  - Jedná sa o dvojrozmernú mapu bitov, ku ktorej sa pristupuje cez súradnice.
  - **Výhody**
    * Programátori-začiatočníci sa často učia programovať cez kreslenie na plátno - môžme predpokladať, že vývojár má skúsenosti s touto formou programovania.
    * Pri pridávaní hry na existujúcu stránku stačí len vytvoriť elementy `<canvas>` na miesto, kde sa má hra na stránke nachádzať a `<script>` so zdrojovým kódom hry.
    * Vývojár je obmedzený na využitie funkcionality plátna a jej rozšírenie cez našu knižnicu, čím dosiahne jednoduchosť, ako vo svojom kóde, tak vo výslednej aplikácii - menej rušivých elementov.
    * Vývoj aplikácie si vyžaduje iba základné znalosti jazyka JavaScript a schopnosť orientovať sa v dokumentácii knižnice.
  - **Nevýhody**
    * Obmedzenie v možnostiach grafického spracovania - je obtiažnejšie pridať napríklad tiene alebo efekty, ktoré sa inak dajú riešiť cez CSS.
    * Optimalizácia rýchlosti chodu aplikácie je presunutá na tvorcu knižnice, prípadne na vývojára aplikácie - animácie môžu byť pomalšie, ak sa vykresľujú stovky elementov.
    * V bežnej praxi sa `<canvas>` využíva iba na zobrazovanie animovaných obrázkov, alebo vizualizáciu dát (grafy), preto má málo vstavanej funkcionality využiteľnej pri tvorbe hier (všetky elementy musíme implementovať my).

∗ Nízka prístupnosť pre ľudí so zdravotnými obmedzeniami.

Keďže pre účely našej knižnice je prioritou simplicita tvorby aplikácií, je vhodné, aby sme otázku zobrazovania aplikácie riešili cez kreslenie na plátno. Dôsledkom obmedzenia možností pri tvorbe aplikácií iba na tie, ktoré poskytuje kreslenie na plátno, je dosiahnutie rovnováhy medzi kontrolou nad prvkami a jednoduchosťou ich implementácie.

Kapitola 2

Návrh knižnice

Tvorba detailného návrhu je prvým krokom k úspešnej implementácii. V tejto kapitole zadefinujeme cieľ, požiadavky a hlavné vlastnosti prvkov knižnice. Podľa toho ďalej navrhneme štruktúru tried objektov, ktoré budeme využívať.

2.1 Cieľ

Edukačné aplikácie, ktoré sa budú použitím našej knižnice vytvárať, budú použité ako jeden z nástrojov pri výučbe. Žiaci k nim budú pristupovať cez zariadenia, ktoré im dá škola k dispozícii - zväčša to sú interaktívne tabule, počítače, alebo tablety [9]. Tieto aplikácie by teda mali fungovať na zariadeniach s rôznymi operačnými systémami a nemali by vyžadovať inštaláciu pred použitím.

Pri tvorbe týchto aplikácií môže programátor vychádzať z predpripravených vzorov, pri čom upravuje len kľúčové aspekty. Tvorca má ale vedieť aj jednoduchým spôsobom vytvoriť novú aplikáciu podľa svojich potrieb, bez väčších obmedzení.

2.2 Požiadavky

Naša knižnica je zameraná na jednoduchú tvorbu edukačných aplikácií, preto si musíme definovať požiadavky, ktoré chceme splniť pri jej implementácii.

- Knižnica má podporovať tvorbu webových aplikácií. Tak sa dá dosiahnuť kompatibilita s rôznymi zariadeniami a absencia nutnosti inštalácie.
- Vytvorené aplikácie musia byť kompatibilné s modernými prehliadačmi. Kompatibilita so staršími verziami, alebo zastaralými prehliadačmi nie je prioritou.
- Aplikácie sa majú dať ovládať ako myšou či klávesnicou, tak i dotykom.

- Knižnica má byť jednoduchá na inštaláciu pri tvorbe aplikácií. Tvorba aplikácií by mala byť možná na rôznych systémoch (Windows/Linux/Mac OS).
- Implementácia knižnice by mala byť jednoduchá aj pre programátora s obmedzenými schopnosťami.
- Knižnica by nemala obmedzovať programátora pri jeho tvorbe. Malo by byť možné v nej tvoriť aj zložitejšie aplikácie.
- Okrem objektov by knižnica mala poskytovať aj funkcie často využívané pri tvorbe aplikácií.
- Tvorca má mať k dispozícii niekoľko vzorových aplikácií, ktoré môže podľa svojej potreby upravovať, alebo z nich vychádzať pri tvorbe nových aplikácií.

2.3 Štruktúra knižnice

Základnou myšlienkou našej knižnice je vytvorenie objektov, ktoré sa zobrazujú a majú nejaké vlastnosti. Tieto objekty musia byť zorganizované do štruktúry tak, aby bolo pri implementácii jasné, ako sa čo robí. Preto budeme pracovať s troma úrovňami objektov:

- **Game** - Objekt hry/aplikácie. Uchováva stav aplikácie a logicky spracúva vstupy od užívateľa.
- **GameElement** - Všeobecný logický objekt. Uchováva svoj stav a dajú sa mu nastaviť jeho interakcie. Predstavuje objekt, s ktorým bude užívateľ môcť manipulovať.
- **GameDrawable** - Všeobecný objekt vykresľovania. Je viazaný na GameElement a dedí jeho stav. Pomocou neho môže byť jeden logický objekt zobrazený ako viacero obrázkov alebo tvarov.

2.4 Hlavná funkcionalita

Pomocou našej knižnice chceme dosiahnuť jednoduchosť tvorby edukačných aplikácií či hier cez zamlčanie mnohých procesov, ktoré by boli inak nevyhnutné implementovať. Takéto procesy alebo funkcie bude knižnica plniť na pozadí, nechávajúc programátora riešiť logickú štruktúru aplikácie. Knižnica by mala zaobstarávať niekoľko často sa opakujúcich vecí, ktoré treba v aplikácii implementovať:

- **Zobrazovanie objektu:** Programátorovi stačí zadefinovať vlastnosti objektu a knižnica zaobstará jeho zobrazenie. Podtriedy triedy GameDrawable budú zabezpečovať zobrazovanie textu, obrázkov, či tvarov.

- **Nastavovanie vlastností objektu:** Objekty GameElement majú vlastnosti ako pozícia či rotácia. Objekty GameDrawable tieto vlastnosti dedia a prinášajú aj vlastné, ako napríklad farba, šírka, výška a tak ďalej. Nastavenie týchto vlastností sa prejaví na zobrazenom objekte.

- **Jednoduchosť inicializácie objektov:** Nutnosť manuálneho prepájania objektov (ako Game s GameElement) je odstránená cez použitie triednych metód na tvorbu objektov. Objekty tiež majú zmysluplné základné nastavenia a ohlásia chybu pri nesprávnej inicializácii (nesprávny typ atribútov a podobne).

- **Reakcia na vstupy od užívateľa:** Objekt Game zachytáva vstupy ako klik, dotyk alebo stlačenie klávesu, ktoré ďalej posúva na spracovanie elementom. Elementy majú metódu pre zistenie, či sa pozícia kliku zhoduje s umiestnením elementu. Niektoré elementy majú preddefinovanú reakciu na vstupy, ale programátor ich vie aj nastaviť ako udalosť.

- **Možnosť nastaviť akciu pri udalosti:** Užívateľ knižnice vie nastaviť, čo sa stane pri udalosti ako je napríklad klik, ťahanie, ukončenie ťahania, držanie tlačidla a ďalších. Takto sa dajú nastaviť udalosti elementom, ale aj objektu Game v prípade, že udalosť nemá byť viazaná na konkrétny element.

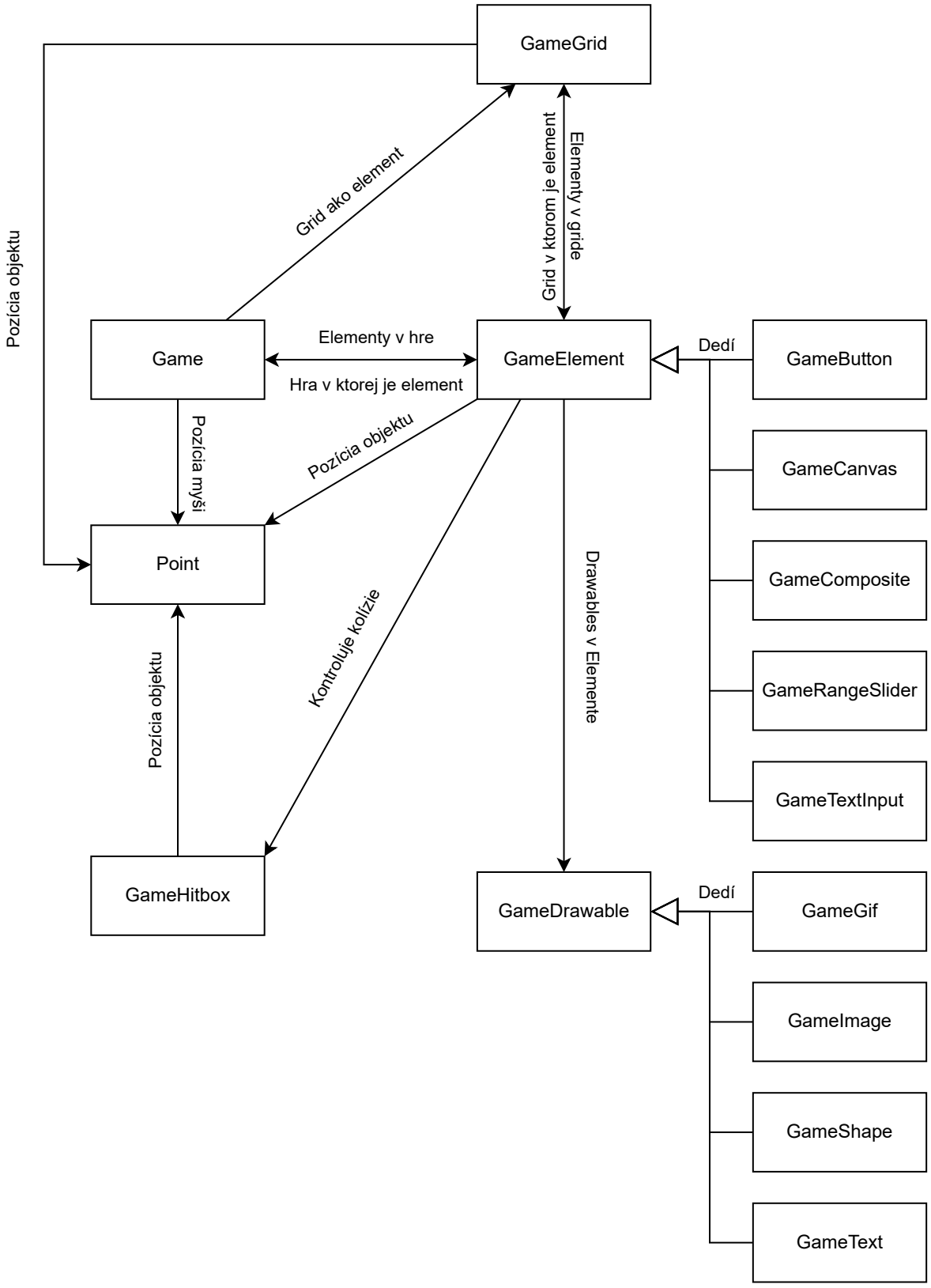
- **Kontrola pozície objektu:** Programátor má niekoľko spôsobov, ako môže sledovať, kde sa nachádza objekt, alebo či sa prekrýva s iným.

  - Elementy si uchovávajú svoju pozíciu, ktorej súradnice sa dajú porovnávať s hodnotami.
  - Programátor môže vytvoriť pomocný element, pre ktorý kontroluje, či sa na ňom nachádza daný bod.
  - Elementy môžu mať zadefinované „hitboxy“. Sú to neviditeľné kruhy previazané s elementmi, pomocou ktorých vie užívateľ kontrolovať kolízie elementov.

2.5 Návrh tried

Okrem vyššie spomenutých všeobecných objektov bude mať vývojár k dispozícii aj ich rozšírenia. Knižnica bude ponúkať takéto triedy:

- **Game**: Hlavný objekt. Uchováva stav aplikácie, spracúva vstupy. Beží na ňom animačná slučka, ktorá vykresľuje objekty v ich momentálnom stave.
- **GameElement**: Všeobecný element. Uchováva svoj stav a vykresľovacie objekty (drawables), ktoré mu patria. Obsahuje metódy na zmenu stavu a priradenie akcie pri udalosti. Triedu GameElement rozširujú tieto triedy:
  - **GameButton**: Špecifický element. Predstavuje obdĺžnikové tlačidlo s textom. Pri stlačení sa spustí vlastná udalosť.
  - **GameCanvas**: Špecifický element. Predstavuje obdĺžnikové plátno, do ktorého sa dá kresliť.
  - **GameTextInput**: Špecifický element. Predstavuje pole na textový vstup. Pri kliku otvorí kontextové okno, do ktorého užívateľ zadá vstupný text. Pri odoslaní vstupu sa spustí vlastná udalosť.
  - **GameRangeSlider**: Špecifický element. Predstavuje element posuvník, ktorému užívateľ môže nastaviť rozmedzie hodnôt. Ťahaním „držadla“ sa mení zvolená hodnota. Pri takejto zmene hodnoty sa spustí vlastná udalosť.
  - **GameComposite**: *Atypický* element. Spravuje viac elementov naraz. Nemá vlastné drawables.
- **GameDrawable**: Všeobecný drawable. Funguje iba ako nadtrieda, nepoužíva sa pri implementácii. Triedu GameDrawable rozširujú tieto triedy:
  - **GameText**: Zobrazuje text.
  - **GameImage**: Zobrazuje obrázok.
  - **GameGif**: Zobrazuje animovaný gif.
  - **GameShape**: Zobrazuje tvar. Tvary môžu byť ovál, štvoruholník, mnohouholník, alebo čiara.
- **GameGrid**: Špeciálny objekt pre štvorcovú sieť, v ktorej môžu byť uložené elementy.
- **GameHitbox**: Špeciálny objekt pre detekciu kolízií elementov.
- **Point**: Objekt predstavujúci pozíciu na súradniciach. Dajú sa pomocou neho robiť operácie ako výpočet vzdialenosti alebo rotácia okolo bodu.



Obr. 2.1: Zjednodušená verzia triedneho diagramu. Podrobný diagram uvádzame v priloženom PDF (príloha 4).

Kapitola 3

Implementácia

V tejto kapitole predstavíme, ako naša knižnica rieši praktické problémy, s ktorými sa programátor môže stretnúť pri tvorbe aplikácie.

3.1 Zobrazovanie elementov

Pri návrhu knižnice sme si zadefinovali tri základné typy objektov, ktoré naša knižnica využíva. Jedná sa o triedy **Game**, **GameElement** a **GameDrawable**.

Naša knižnica vykresľuje pole aplikácie na HTML element `<canvas>`. Na to, aby sa správne zobrazovali animácie či presúvajúce objekty, treba neustále opakovať vykresľovanie plátna. Okrem toho, že opakované kreslenie na plátno je potrebné pri zmene pozícii, tvaru či atribútov elementov, je dôležité obnovovať stav plátna aj pre animované obrázky (GIFy).

V tejto časti si vysvetlíme, ako naša knižnica delí úlohy spojené s vykresľovaním jej triedam.

3.1.1 Trieda GameDrawable

Trieda **GameDrawable** je abstraktná trieda, ktorá predstavuje rozhranie pre triedy **GameGif**, **GameImage**, **GameShape** a **GameText**. Každá z týchto tried sa stará o kreslenie iného typu objektu, ako sú obrázky či text. Čo a ako budú jednotlivé inštancie tried kresliť, môže programátor nastaviť cez ich atribúty.

Rozhranie **GameDrawable** poskytuje základné vlastnosti, ktoré jej potomkovia zdieľajú. Jedná sa o tieto atribúty:

- *name* - meno objektu. Používa sa pri získavaní referencie na neho od objektu typu **GameElement**, v ktorom sú drawables uložené.

- *level* - hĺbka vykresleného objektu v rámci elementu. Objekty s menšou hodnotou sa vykreslia pod tými s väčšou.
- *dx, dy* - deviácia od stredu elementu.
- *width, height* - rozmery kreslených objektov.
- *rotation* - uhol otočenia objektu v radiánoch.
- *visible* - viditeľnosť objektu. Objekt sa zobrazí, iba keď je *visible* nastavené na `true`.
- *hScale, vScale* - škála kreslených objektov. Pomocou týchto atribútov sa dajú objekty zväčšovať, zmenšovať, či zrkadliť (nastavenie záporných čísel).

Táto trieda má definované dve dôležité metódy súvisiace s kreslením na plátno:

- `draw(ctx,center)` - metóda na nastavenie plátna a vykreslenie objektu. Nastaví vlastnosti kontextu plátna (*ctx*) posunutého ako parameter (rotácia, pozícia na plátne) a zavolá metódu `drawFunction` na kreslenom objekte. Po vykreslení objektu na plátno vráti kontextu pôvodné nastavenia.
- `drawFunction(ctx)` - abstraktná metóda. Triedy rozširujúce triedu **GameDrawable** pomocou nej kreslia objekt na plátno.

Trieda GameText

Trieda **GameText** zaobaľuje zobrazenie textu na plátne. Zobrazenie je okrem zdedených ovplyvnené aj týmito atribútmi, špecifickými pre triedu.

- *color* - farba textu. Je to string vo formátoch `"blue"` (mená farieb definované pre CSS), `"##11FF11"` (hexadecimálne hodnoty farby), alebo `"random"`, ktoré s použitím funkcie `randomColor()`, ktorú dáva knižnica k dispozícii, vygeneruje náhodnú farbu v hexadecimálnom formáte.
- *text* - zobrazený text.
- *font* - veľkosť a font písma. Predvolená hodnota je `"20px arial"`.
- *maxWidth* - maximálna šírka textu. Pomocou tohto atribútu sa dá dlhý text zúžiť, aby sa napríklad zmestil do obdĺžnika.

Pomocou tejto triedy vieme zobraziť iba jeden riadok textu. V prípade potreby zobrazenia viacerých riadkov je treba elementu priradiť viac inštancií tejto triedy a využiť atribút *dy* zdedený z nadtriedy.

Trieda GameImage

Táto trieda zaobstaráva zobrazovanie obrázkov na plátne.

Okrem zdedených má iba jeden atribút - *img*, v ktorom je uložená referencia na zobrazený obrázok. Zobrazený obrázok sa môže po inicializácii zmeniť pomocou metódy `setImg(imageName)`. Súbory s obrázkami musia byť vždy v priečinku `resources/`.

Trieda GameGif

Táto trieda sprístupňuje zobrazovanie animovaných obrázkov (GIFov).

Zobrazovanie animovaných GIFov nie je triviálne. Keďže sa nejedná o statický obrázok, ale o sériu obrázkov, ktoré sa za sebou zobrazujú v slučke, užívateľ knižnice najskôr musí uložiť obrázky vo formáte, ktorý môže knižnica využiť. Na túto potrebu sme vytvorili pythonovský skript (príloha 3), ktorý z animovaného GIFu `menoSuboru.gif` vytvorí dva súbory - `menoSuboru_sheet.png`, ktorý predstavuje rozprestretý GIF (Obr. 3.1) a `menoSuboru_data.json`, v ktorom sú údaje ako výška, šírka obrázku a počet snímok GIFu.

Obr. 3.1: Ukážka rozprestretého GIFu.

Pri inicializácii, alebo neskôr s použitím metódy `setImg`, sa objektu posunie meno GIFu (bez extenzie .gif). Trieda si načíta dáta zo súborov vytvorených vyššie spomínaným skriptom a používa ich pri zobrazovaní.

Trieda **GameGif** má takéto atribúty:

- *img* - referencia na obrázok (`gif_sheet.png`).
- *imgData* - načítané dáta o rozmere obrázku a počte snímok (súbor `gif_data.json`).
- *stagger* - počet preskočených obnovení obrazovky. Ak programátor nastaví vyššie číslo, spomalí to animáciu GIFu.
- *currentFrame* - poradové číslo momentálne zobrazenej snímky.

Dôležitá metóda pre túto triedu je metóda `updateAnimation()`, ktorá zabezpečuje striedanie snímok. Túto metódu volá trieda **GameElement** pri každom vykreslení objektu.

Trieda GameShape

Táto trieda zabezpečuje zobrazenie štyroch typov tvarov: oval, rectangle, line a polygon. Keďže tieto tvary sú definované rôznymi parametrami, trieda ponúka takéto atribúty:

- *type* - typ tvaru. Je to jeden z `["rectangle","oval","polygon","line"]`.
- *fill* - farba výplne tvaru. Podobne ako pri farbe textu to je string s menom farby, hexadecimálna hodnota alebo `"random"`.
- *stroke* - farba čiary. Formát rovnaký ako pri výplni.
- *lineWidth* - šírka čiary.
- *rx, ry* - polomer oválu na X-ovej a Y-ovej osi.
- *coords* - súradnice bodov na čiare alebo mnohouholníku. Je to zoznam čísel párnej dĺžky.

Kľúčové atribúty pre *rectangle* sú *width* a *height*, *oval* je definovaný atribútmi *rx, ry* a tvary *line* a *polygon* sú definované zoznamom *coords*.

Keďže v niektorých situáciách je potrebné meniť tvar za behu, definovali sme aj metódy `setLine(from,to)` a `addPoint(point)`, ktoré využívajú inštancie nami definovanej triedy **Point** ako parametre. Tieto metódy upravujú stav atribútu *coords*. Metóda `setLine` ho nastaví na dĺžku 4 (čiara definovaná dvoma bodmi) a metóda `addPoint` pridá súradnice bodu na koniec zoznamu.

3.1.2 Trieda GameElement

Trieda **GameElement** predstavuje vizuálno-logický blok. Uchováva si referencie na drawables a pri volaní metódy `draw(ctx)` ich postupne vykreslí na plátno. Okrem volaní funkcie na vykresľovanie drawables sa trieda stará aj o animáciu GIFov pomocou metódy `animate()`.

Táto trieda má veľa nastaviteľných atribútov a niektoré z nich súvisia s vykresľovaním:

- *level* - úroveň elementu. Podobne ako drawables, elementom sa dá nastaviť úroveň, na ktorej sa vykreslia (nižšie číslo značí, že sa vykreslia pod elementami s vyššou hodnotou). Rozdiel je v tom, že úroveň elementu sa porovnáva s úrovňou iných elementov v rámci inštancie triedy **Game**, do ktorej patria, pričom úroveň drawables sa porovnáva s úrovňou ostatných drawables v rámci konkrétneho elementu.

- *center* - absolútna pozícia elementu na plátne. Od tejto pozície sa počíta deviácia pri objektoch **GameDrawable**. Atribút *center* je typu **Point**, ktorý je definovaný hodnotami x a y. Jeho hodnoty sa dajú meniť rôznymi spôsobmi:
  - priradenie novej inštancie cez `element.center = new Point(x,y)`,
  - zmena hodnôt atribútu napríklad ako `element.center.x = 10`,
  - zmena hodnôt priamo cez `element.x = 10`,
  - využitie metódy `element.setPosition(x,y)`.
- *children* - zoznam inštancií typu **GameDrawable**, ktoré sú priradené elementu.
- *rotation* - uhol rotácie. Okrem rotovania jednotlivých drawables vieme rotovať aj celý element.
- *visible* - viditeľnosť elementu. Podobne ako pri drawables sa dá vypnúť zobrazovanie elementu cez nastavenie tohto atribútu na `false`.

Funkcionalita triedy **GameElement** je pomerne široká. Do vykresľovania zasahujú nasledujúce metódy:

- Vyššie spomínané metódy, ktoré menia pozíciu elementu, napríklad metóda `element.setPosition(x,y)`.
- Metódy, ktoré vytvárajú inštancie drawables a priraďujú ich do elementu:
  - `element.createGif(gifName,attrs)`
  - `element.createImage(imageName,attrs)`
  - `element.createShape(type,attrs)`
  - `element.createText(text,attrs)`
- Metóda `element.draw(ctx)`, ktorá nastaví vlastnosti kontextu plátna (ctx) ako je rotácia a pozícia elementu, posunutého z objektu **Game** a postupne volá metódy `drawable.draw(ctx)` pre každý drawable objekt zo zoznamu v triednom atribúte *children*. Do tejto metódy posiela kontext plátna a referenciu na svoj stred (atribút *center*) ako parameter.
- Metóda `element.animate()`, ktorá zavolá metódu `updateAnimation()` všetkým priradeným objektom typu **GameGif**.

Podtriedy triedy GameElement

Trieda **GameElement** má aj svoje podtriedy, ktoré dedia jej vlastnosti, ako je napríklad trieda **GameButton**. Tieto podtriedy nemajú žiadne vlastné špecifické metódy na vykresľovanie. Líšia sa v tom, že majú preddefinované drawables, ktoré im patria a preddefinované akcie pri konkrétnych vstupoch (eventoch). V prípade triedy **GameButton** sa jedná o objekty predstavujúce obdĺžnik (**GameShape**) a zobrazený text (**GameText**).

3.1.3 Trieda Game

Trieda **Game** ukladá referenciu na zobrazený HTML element `<canvas>` a jeho kontext (využívaný pri kreslení). Tiež si ukladá referencie na priradené elementy do zoznamu v triednom atribúte *elements*. Tento zoznam sa udržuje zoradený podľa atribútu *level*, aby sa elementy vykresľovali v správnom poradí.

Po inicializácii triedy sa spustí slučka, ktorá každých 30 milisekúnd obnoví stav plátna. Pri každom obnovení stavu plátna sa najskôr vyčistí plátno a zavolajú sa metódy `element.draw(ctx)` a `element.animate()` pre každý priradený element.

Vykresľovacia slučka sa dá kedykoľvek zastaviť a znovu spustiť použitím metód `stopAnimation()` a `animate()`.

3.2 Detekcia kliku na element

Aplikácie vytvorené v našej knižnici sa dajú ovládať myšou, dotykom, alebo klávesnicou. Programátor vie nastaviť, ako sa elementy budú pri vstupe správať. Najskôr ale treba vyriešiť, ako program vôbec zistí, že užívateľ klikol na daný element.

Keď nastane akcia „klik“, program vie, na akých súradniciach sa udiala. Avšak zistenie, na ktorý element užívateľ klikol, nie je triviálne. Je niekoľko spôsobov, ako sa to dá zistiť:

- **Výpočet vzdialenosti od stredu elementu**. Program prejde zoznamom všetkých elementov a porovnáva vzdialenosť od ich stredu k pozícii myši. Pri nájdení elementu so vzdialenosťou od myši v nastavenom rozsahu, sa zvolí.
  NEVÝHODA: Klikanie na objekty v inom tvare ako kruh môže byť neintuitívne. Takisto si to vyžaduje nadbytočné nastavovanie takýchto vzdialeností pre každý element.

- **Neviditeľný obdĺžnik okolo elementu**. Program prejde zoznamom všetkých elementov a kontroluje, či sú súradnice myši v prednastavenom rozsahu pre element. Ak je myš vo vnútri tohto rozsahu, element sa zvolí.
  NEVÝHODA: Pri rotácii elementov musíme rotovať aj kliknutý bod. Podobne ako v predchádzajúcom prípade si to tiež vyžaduje nastavovanie oblasti a klikanie na objekty v inom tvare ako obdĺžnik môže byť neintuitívne.

- **Kontrola transparencie pixelu pod myšou**. Program prejde zoznamom všetkých elementov zoradených podľa úrovne (*level*) od najvyššieho a vykresľuje ich na (neviditeľné) plátno. Pre každý vykreslený element kontroluje, či je kliknutý pixel transparentný. Ak nie je transparentný, zvolí taký element. Takto aj keď sa elementy prekrývajú, dá sa kliknúť na spodný, pokiaľ aspoň trochu pretŕča. Takýmto spôsobom sa dá zvoliť element cez klikanie priamo na jeho vykreslené časti.
  NEVÝHODA: Je to trochu pomalšie.

Pre potreby našej knižnice sme zvolili tretí spôsob.

Trieda **Game** obsahuje metódu `getElementAtPos(position)`, ktorá sa volá pri každom kliknutí na plátno. Táto metóda postupne volá metódu `isInside(position)` na každom elemente počnúc od najvyššieho (podľa atribútu *level*). Ďalší postup je podobný ako pri vykresľovaní objektov. Najskôr sa vyčistí kontext plátna, potom sa rotuje a posúva podľa nastavených atribútov elementu. Element potom vykreslí všetky svoje drawables, rovnako ako pri funkcii `draw`. Po vykreslení kontroluje, či je pixel na pozícii kliku transparentný. Ak nie je, vráti hodnotu `true`, čo ukončí proces detekcie elementu na pozícii.

3.3 Detekcia kolízií objektov

V edukačných aplikáciách je často potrebné porovnávať pozície viacerých elementov. Napríklad v ukážke na Obr. 1.3 program kontroluje, či sa presúvaná žabka nachádza na váhach. Ak je na správnej pozícii, program zmení svoj stav, inak sa žabka vráti na svoje pôvodné miesto.

Naša knižnica ponúka tri spôsoby, ktorými sa dá pozícia kontrolovať. Všetky sú rovnako validné a majú svoje miesto podľa uváženia programátora.

1. **Porovnávanie pozície stredu elementu so súradnicami**. Programátor nastaví rozsah súradníc a pri prevedení akcie nad elementom sa s týmto rozsahom porovnáva pozícia stredu elementu.

2. **Využitie funkcie `isInside` iného elementu**. Programátor vytvorí cieľový element, do ktorého chce, aby užívateľ presunul daný objekt. Program potom pomocou funkcie `cielovyElement.isInside(pozicia)` porovnáva, či sa pozícia (stred) manipulovaného elementu nachádza na cieľovom elemente.

3. **Využitie objektu GameHitbox**. Ku každému elementu sa dá pridať hitbox, ktorý predstavuje kružnicu definovanú jej polomerom a deviáciou od stredu elementu. Programátor pomocou týchto objektov vie zistiť, kedy sa objekty prekrývajú.

Prvé dva spôsoby využívajú základné vlastnosti elementov, ako je ich pozícia a detekcia kliku na nich (metóda `isInside`). Posledný spôsob funguje využitím metódy `element.collidesWith(other)` (Kód 1), ktorá pracuje s objektami **GameHitbox**.

```
collidesWith(other) {
    if (this.hitboxes.length === 0) {return false}
    if (other.hitboxes.length === 0) {return false}
    for (const hb1 of this.hitboxes) {
        const pos1 = this.center
                        .add(hb1.delta)
                        .rotateAround(this.center,this.rotation)
        for (const hb2 of other.hitboxes) {
            const pos2 = other.center
                            .add(hb2.delta)
                            .rotateAround(other.center,other.rotation)
            const distance = pos1.distanceTo(pos2)
            if (distance < hb1.r + hb2.r) {return true}
        }
    }
    return false
}
```

Kód 1: Ukážka metódy GameElement.collidesWith(other)

Trieda **GameHitbox** si uchováva svoju pozíciu vo forme atribútu *delta* triedy **Point**, ktorý predstavuje deviáciu od stredu elementu, ku ktorému je hitbox priradený. Tiež je definovaná polomerom kruhu (atribút r), ktorého prekrytie s iným hitboxom sa vypočítava.

V kóde 1 vidíme prácu s inštanciami triedy **Point** (atribút *center* v elemente a atribút *delta* v hitboxe). Porovnávame pozície a polomery hitbxov medzi elementami. Najskôr zoberieme stred elementu a pripočítame k nemu deviáciu od neho uloženú v hitboxe. Tento bod ďalej rotujeme okolo stredu elementu o uhol *rotation*. Keď toto spravíme pre oba porovnávané hitboxy, porovnáme vzdialenosť vypočítaných bodov so súčtom polomerov kruhov, ktoré hitboxy predstavujú. Ak sa prekrývajú, teda ich vzdialenosť je menšia ako súčet ich polomerov, funkcia vráti `true`, čo znamená, že elementy sa prekrývajú.

Túto funkciu môže programátor využiť, ak chce kontrolovať prekrytie dvoch elementov, ale často sa stáva, že programátor chce zistiť, s akým elementom sa daný element prekrýva. Na to slúži metóda `checkCollisions` (kód 2) definovaná v triede **Game**.

```
checkCollisions(element) {
    const collisions = []
    for (const other of this.elements) {
        if (element === other) {
            continue
        }
        if (element.collidesWith(other)) {
            collisions.push(other)
        }
    }
    return collisions
}
```

Kód 2: Ukážka metódy Game.checkCollisions(element)

Metóda `checkCollisions` prejde zoznamom elementov a volá pre vstupný element metódu `collidesWith` so všetkými ostatnými elementami. Tie, ktoré sa prekrývajú so vstupným elementom, sa pridajú do zoznamu, ktorý metóda vráti.

3.4 Eventy

Na to, aby bola aplikácia interaktívna, musí vedieť reagovať na vstupy od užívateľa. Tieto vstupy môžu byť rôzneho typu, ako napríklad klikanie, ťahanie, držanie myši, stlačenie klávesu a podobne. Takéto vstupy označujeme ako eventy. Odchytávanie niektorých eventov zabezpečuje samotný jazyk JavaScript, ale implementácia iných, ako napríklad ťahanie objektu myšou, nie je triviálna. Preto naša knižnica definuje vlastné eventy a ponúka nástroje na ich spracovanie.

Keďže štruktúra našej aplikácie je orientovaná na využitie elementov ako logické bloky, programátor by im mal vedieť nastaviť, akú akciu vykonajú pri odchytení eventu.

Trieda **GameElement** je základným elementom, z ktorého ostatné triedy, ako napríklad trieda **GameButton**, dedia jej vlastnosti a stavajú na nich. Pre každý event, ktoré si neskôr definujeme, knižnica ponúka metódy `element.add...Listener()` a `element.remove...Listener()`, kde tri bodky predstavujú meno eventu. Programátor vytvorí funkciu, ktorá sa vykoná pri odchytení eventu, a použitím vyššie spomínaných metód ju previaže so žiadaným eventom.

Použitie eventov demonštrujeme na príklade (Kód 3) - vytvoríme si štvorec, ktorého farba sa bude meniť pri každom kliku na neho:

```
const element = game.createElement({clickable:true})
element.createShape("rectangle",{width:200,height:200,name:"rect"})

function vykonajPriKliku(event) {
    const stvorec = this.getChildByName("rect")
    stvorec.fill = "random"
}
element.addOnClickListener(vykonajPriKliku)
```

Kód 3: Ukážka využitia eventov ponúkaných knižnicou.

Funkcie, ktoré sa posielajú ako parameter do metódy `add...Listener()`, môžu využívať kľúčové slovo `this`, ktoré predstavuje element reagujúci na event. Alternatívne, namiesto použitia metódy `getChildByName` na získanie prepojeného drawable, by sme si mohli uložiť referenciu na štvorec pri jeho inicializácii a vo funkcii sa na ňu odkazovať. Využitie metódy `getChildByName` je však vhodné, ak pracujeme s viacerými elementami, ktorých drawables chceme meniť.

V nasledujúcich podčastiach predstavíme eventy, ktoré naša knižnica ponúka programátorovi.

3.4.1 Eventy pre triedu GameElement

Trieda **GameElement** predstavuje element, s ktorým bude programátor pracovať asi najviac. Špecifické elementy, ako napríklad **GameButton**, dedia vlastnosti triedy **GameElement**, čo sa týka aj nástrojov pre spracovanie eventov.

Trieda **GameElement** sa stará o spracovanie týchto eventov:

- *onClick* - nastane pri kliku na element.
- *onDrag* - nastane pri hýbaní elementu myšou.
- *onStartDragging* - nastane pri kliku, podobne ako *onClick*, ale iba vtedy, keď je element posúvateľný.
- *onFinishDragging* - nastane pri pustení posúvaného elementu.
- *onMouseHold* - nastane v pravidelných intervaloch (každých 50ms) pri držaní elementu.
- *onFinishMouseHold* - nastane pri pustení držania elementu myšou.
- *onKeyPress* - nastane pri stlačení tlačidla na klávesnici.
- *onKeyHold* - nastane v pravidelných intervaloch pri držaní tlačidla na klávesnici.
- *onKeyUp* - nastane pri pustení tlačidla na klávesnici.
- *onMove* - nastane pri volaní funkcie `element.move(delta)` a pri hýbaní elementom myšou.

Elementom sa dajú nastaviť atribúty, ktoré povoľujú či zakazujú odchytávanie eventov, či akcie s tým spojené. Tieto atribúty sú nasledujúce:

- *clickable* - nastavenie na `true` povoľuje využitie eventov spojených s klikaním.
- *draggable* - nastavenie na `true` povoľuje presúvanie elementu a tiež využitie eventov s tým spojených.
- *pressable* - nastavenie na `true` povoľuje využitie eventov spojených so stláčaním klávesov.
- *holdable* - nastavenie na `true` povoľuje využitie eventov spojených s držaním (myšou).
- *stationary* - nastavenie na `true` zabezpečuje, že element, ktorý reaguje na posúvanie myšou, ostane na svojom mieste. Konkrétny prípad využitia tejto vlastnosti je napríklad slider (**GameRangeSlider**), ktorého časť sa dá presúvať, ale je kľúčové, aby celý objekt nemenil svoju pozíciu.

3.4.2 Eventy pre triedu GameButton

Trieda **GameButton** je rozšírením triedy **GameElement**. Predstavuje tlačidlo, ktoré reaguje na stlačenie, držanie a pustenie myši.

Okrem eventov, ktoré dedí z triedy **GameElement**, dáva programátorovi k dispozícii špecifický event *onPress*. Event *onPress* sa líši od eventu *onClick* tým, že nastane, keď užívateľ pustí myš, pričom jej pozícia je na tlačidle. Tento prístup dáva užívateľovi možnosť si rozmyslieť, či chce tlačidlo stlačiť, aj keď už ho začal držať.

Tlačidlo sa pri držaní myšou zvýrazní, tak užívateľ vidí, ktoré z tlačidiel (ak ich je viac) má označené.

3.4.3 Eventy pre triedu GameRangeSlider

Trieda **GameRangeSlider** je rozšírením triedy **GameElement**. Predstavuje posúvateľný slider, ktorý si uchováva hodnotu medzi *min* a *max*, ktoré mu užívateľ nastaví.

Okrem eventov, ktoré dedí z triedy **GameElement**, dáva programátorovi k dispozícii špecifický event *onChange*. Event *onChange* sa vykoná pri každej zmene stavu slidera. Užívateľ môže kedykoľvek zavolať metódu `slider.getValue()`, ktorá vráti hodnotu medzi *min* a *max*.

3.4.4 Eventy pre triedu GameTextInput

Trieda **GameTextInput**, predstavújúca textové pole, je rozšírením triedy **GameElement**. Keďže elementy sa vykresľujú na plátno, museli sme nejako vyriešiť to, ako sa bude dať v aplikácii zadávať text. Na tento účel sme vytvorili triedu **GameTextInput**, ktorá pri kliku otvorí kontextové okno, do ktorého sa dá zadať text.

Trieda **GameTextInput** dáva užívateľovi k dispozícii špecifický event *onEnter*. Event *onEnter* sa vykoná pri potvrdení zadaného textu do okna. Užívateľ tak môže nastaviť, aká akcia sa vykoná po zadaní textu.

3.4.5 Eventy pre triedu Game

V určitých situáciach programátor nechce, aby boli akcie vykonané po evente viazané na konkrétny element. Pre takéto situácie odchytáva eventy aj samotná trieda **Game**. Eventy ktoré jej definujeme, sú nasledovné:

- *onClick* - nastane pri stlačení tlačidla na myši,
- *onMove* - nastane pri každom pohybe myši,
- *onMouseUp* - nastane pri pustení tlačidla na myši,
- *onClear* - nastane pri zavolaní funkcie `game.clear()`.

Keď chceme, aby aplikácia reagovala rovnako na vstupy z myši ako na vstupy z dotyku, je výhodné využiť eventy definované v tejto triede namiesto využitia eventov na objekte `<canvas>`. Takto predídeme nadbytočnej duplicite a prípadným problémom s kompatibilitou s mobilnými zariadeniami.

3.5 Presúvanie objektov myšou

V edukačných aplikáciách sa často vyskytuje chytanie a presúvanie objektov myšou. Použitím čistého JavaScript-u sa to väčšinou robí rozdelením procesu na 3 časti:

1. Prvý klik - program zachytí eventy *mousedown* a *touchstart*. Pri zachytení uloží presúvaný objekt do premennej.

2. Pohyb myši - program pri každom pohybe myšou (alebo dotykom) zachytáva eventy *mousemove* a *touchmove*. Ak je zvolený objekt, ktorý sa má presúvať, zmenia sa jeho súradnice podľa pozície myši.

3. Ukončenie držania myši - program zachytí eventy *mouseup* a *touchend*. Pri zachytení program nastaví premennú na prázdnu hodnotu a tak ukončí presúvanie objektu.

Keďže tento proces je často opakovaný a pomerne zdĺhavý, je vhodné pridať posúvateľnosť ako vlastnosť objektu.

Trieda **Game** odchytáva vyššie spomínané eventy a spracúva ich v troch samostatných metódach.

- Metóda `mouseDown(event)` - zavolá funkcie pre všeobecný event *onClick* neviazaný na element. Ak užívateľ klikol na klikateľný element (podľa atribútu *clickable*), zavolá na ňom metódu `click`, ktorá zavolá všetky listenery na klik elementu. Ak je element posúvateľný alebo držateľný (atribúty *draggable* a *holdable*), uloží elementy do členskej premennej *selectedElement*. Ak je presúvateľný, trieda **Game** si uloží aj deviáciu kliknutej pozície od stredu zvoleného elementu do členskej premennej *selectedDelta* (aby užívateľ pri presúvaní držal element na tom istom mieste, na ktoré klikol) a zavolá na elemente metódu `startDragging`, ktorá zavolá všetky listenery na začiatok posúvania elementu. Ak je držateľný, zavolá na ňom metódu `startMouseHold`, ktorá spustí slučku, ktorá periodicky volá listenery na tento event.

- Metóda `mouseMove(event)` - zavolá funkcie pre všeobecný event *onMove* neviazaný na element. Ak je zvolený element (členská premenná *selectedElement*) posúvateľný, zavolá na ňom metódu `drag`, ktorá zmení jeho pozíciu a zároveň volá listenery pre eventy *onMove* a *onDrag*.

- Metóda `mouseUp(event)` - zavolá funkcie pre všeobecný event *onMouseUp* neviazaný na element. Ak je zvolený element posúvateľný, zavolá na ňom metódu `finishDragging`, ktorá zavolá listenery priradené eventu na tom elemente. Ak je držateľný, zavolá na ňom metódu `finishMouseHold`, ktorá zastaví slučku pre event *onMouseHold* a zavolá listenery pre event *onFinishMouseHold*. Nakoniec nastaví členské premenné *selectedElement* a *selectedDelta* na prázdnu hodnotu a tým ukončí presúvanie elementu.

```
drag(mousePos,delta,event) {
    if (!this.stationary) {
        this.center = mousePos.subtract(delta)
        for (const callback of this.onMove) {
            callback.call(this,event)
        }
    }
    for (const callback of this.onDrag) {
        callback.call(this,event)
    }
}
```

Kód 4: Ukážka metódy GameElement.drag(mousePos,delta,event)

V kóde 4 vidíme telo metódy `element.drag()`, ktorá sa volá pri pohybe myšou, ak je označený posúvateľný element. Metóda dostane pozíciu myši a jej deviáciu od stredu elementu ako parametre typu **Point**. Nastaví elementu pozíciu na rozdiel týchto dvoch hodnôt (tak, aby bola myš prichytená na tom istom mieste, ako pri prvom kliku). Nakoniec zavolá listenery (funkcie) pre eventy *onMove* a *onDrag*.

Parameter *event* predstavuje event posunutý od plátna. Je to objekt, ktorý obsahuje informácie ako absolútna pozícia myši a stlačené tlačidlo (pravé, ľavé, stredné). S týmito informáciami potom môže programátor pracovať pri definícii listenerov na eventy (napríklad nastavenie akcie pri stlačení pravého či ľavého tlačidla).

Na zistenie pozície myši (alebo dotyku) na plátne z parametra *event* slúži metóda `game.getMousePos(event)` (Kód 5). Metóda ukladá zistenú hodnotu do atribútu *shared*, ktorý je zdieľaný medzi objektom **Game** a jemu priradenými elementami. To umožňuje užívateľovi knižnice pristupovať k pozícii myši priamo cez objekt `element.shared.mousePos`, bez nutnosti volania metódy `game.getMousePos(event)`.

Atribút *shared* okrem pozície myši obsahuje aj referenciu na kontext neviditeľného plátna, ktoré sa využíva pri volaní metódy `isInside` (spomínaná v časti 3.2).

```
getMousePos(event) {
    const rect = this.canvas.getBoundingClientRect();
    let e = event
    if (!(event instanceof MouseEvent)) {
        // TouchEvent not defined in desktop browser
        e = event.touches.item(0)
        if (e === null) {return undefined}
    }
    this.shared.mousePos = new Point(
        e.clientX - rect.left,
        e.clientY - rect.top
    )
    return this.shared.mousePos
}
```

Kód 5: Ukážka metódy Game.getMousePos(event)

3.6 Kompozit elementov

V niektorých situáciách chceme manipulovať s viacerými elementami naraz. Môžme to riešiť napríklad tak, že presunieme všetky drawables z viacerých elementov do jedného. Voliť však tento spôsob často nie je optimálne, napríklad v situácii, keď chceme neskôr elementy rozdeliť a manipulovať s nimi samostatne. Pre túto potrebu sme vytvorili triedu **GameComposite**, ktorá zjednodušuje riešenie týchto situácií.
Trieda **GameComposite** predstavuje pseudo-element, ktorý nemá žiadne drawables, ale namiesto toho si ukladá zoznam priradených elementov. Elementy sa pri vložení do kompozitu odstránia z inštancie triedy **Game**. Priradeným elementom je odobraná schopnosť reagovať na eventy a preberá to za nich kompozit.

Trieda plní svoje funkcie nasledovne:

- **Vykresľovanie elementov** - Kompozit prepisuje funkcie `draw` a `animate` triedy **GameElement**. Volá tieto funkcie na každom elemente, ktorý mu je priradený.
- **Pozícia** - Priradené elementy si zachovávajú svoju pozíciu. Pozícia stredu kompozitu sa dá bez zmeny pozície priradených elementov nastaviť pomocou metódy `setCenter`. Pri nastavení pozície kompozitu (metóda `setPosition` alebo priradenie novej hodnoty pre atribút *center*) sa zmení aj pozícia priradených elementov.

- **Detekcia kliku** - Kompozit prepisuje funkciu `isInside` zdedenú od triedy **GameElement** tak, že namiesto volania funkcie v drawables ju volá v priradených elementoch.
- **Kolízie** - Podobne ako v predchádzajúcom bode, kompozit prepisuje metódu `collidesWith` tak, že kontroluje kolízie jednotlivých elementov.
- **Rotácia kompozitu** - Atribút *rotation* v triede **GameElement** mení nastavenia kontextu plátna. Keďže kompozitu sú priradené elementy, ktorých pozícia sa musí meniť, nestačí len rotovať plátno, ale treba aj meniť pozíciu elementov. Na tieto účely sme vytvorili metódu `rotateElements` (Kód 6), ktorá nahradzuje využitie atribútu *rotation*.
- **Pridávanie a odstraňovanie elementov** - Pri pridávaní elementu mu sú nastavené atribúty *clickable*, *draggable* a *pressable* na `false` a odstráni sa referencia na element z triedy **Game**. Pôvodné hodnoty atribútov sa ukladajú do spoločného objektu v triednom atribúte *elements*. Pri odstraňovaní sa elementom vrátia ich pôvodné nastavenia a element sa opäť pridá do zoznamu *elements* v **Game**.

```
rotateElements(origin,angle,keepOrientation=false) {
    for (const element of this.elements.map(e=>e.element)) {
        element.center = element.center.rotateAround(origin,angle)
        if (!keepOrientation) {
            if (element instanceof GameComposite) {
                element.rotateElements(element.center,angle,false)
            } else {
                element.rotation += angle
            }
        }
    }
}
```

Kód 6: Ukážka metódy GameComposite.rotateElements()

Na ukážke kódu 6 vidíme, ako funguje rotácia kompozitom. Zvolíme si bod *orgin*, okolo ktorého budeme rotovať elementy o uhol *angle*. Ostatné elementy rotujeme okolo bodu *origin*, využívame pri tom metódu `rotateAround` triedy **Point**. Ak chceme, aby uhol rotácie elementu ostal nezmenený, nastavíme parameter *keepOrientation* na `false`. Pre každý element, ktorý je kompozitom, voláme túto metódu rekurzívne.

3.7 Inštalácia

Knižnica sa dá inštalovať dvoma spôsobmi:

1. Skopírovať priečinok `easy-educational-games/modules/` niekam do aplikácie a triedy importovať priamo zo súboru `modules/index.js`.

2. Využiť Node.js a k nemu pribalený nástroj npm.

Hlavnou výhodou využitia nástroja npm je jednoduchosť udržovania aktuálnej verzie knižnice.

Najjednoduchší spôsob, akým môže užívateľ začať využívať našu knižnicu, je stiahnuť si vzorový projekt (príloha 2) a nainštalovať potrebné balíky pomocou príkazu `npm install` (podrobnejší návod na inštaláciu je na stránke vzorového projektu).

Nástroj Node.js presúva beh kódu z klienta na server, preto pri tomto spôsobe riešenia musíme využiť balík Express, ktorý využívame na presunutie kódu naspäť do prehliadača.

Kroky na zakomponovanie našej knižnice do nového projektu sú takéto:

1. Otvoríme si príkazový riadok a presunieme sa do priečinku, v ktorom bude aplikácia.

2. Príkaz `npm init -y` - vytvorí súbor `package.json`.

3. Príkaz `npm install easy-educational-games --save` - stiahne knižnicu z internetu.

4. Vytvoríme si súbor „index.js“ a vložíme do neho tento kód:

```
const g = require("easy-educational-games")
const path = require("path");
//set static folder
g.app.use(g.express.static(path.join(__dirname,"public")))
g.app.listen(g.PORT,"0.0.0.0",
    ()=>console.log(`Server running on port ${g.PORT}`)
)
```

   Tento kód zabezpečí vytvorenie cesty k modulom pre prehliadač a spustenie servera.

5. Vytvoríme si priečinok public, v ktorom bude časť spúšťaná prehliadačom - v ňom vytvoríme súbory „index.html“, „script.js“ a priečinok „resources“, do ktorého patria obrázky, ktoré budú súčasťou aplikácie.

6. Súbor „index.html“ je stránka aplikácie. V ňom je dôležité vytvoriť element `<canvas id="game"></canvas>`, ktorý predstavuje okno aplikácie a ďalej element `<script type="module" src="script.js"></script>`, ktorý prepojí kód v súbore „script.js“ s touto stránkou. Na poradí záleží, element `<script>` patrí na koniec súboru.

7. Do súboru „script.js“ vložíme hlavičku:

```
import {Game} from "/modules/index.js"
const canvas = document.getElementById('game')
canvas.width = 600
canvas.height = 600
const game = new Game(canvas)
```

   Takto prepojíme knižnicu so súborom, ktorý predstavuje zdrojový kód aplikácie, nastavíme rozmery plátna a vytvoríme inštanciu hry (**Game**), pomocou ktorej vytvárame a pracujeme s objektami.

8. Server spustíme cez príkaz `node index.js`. V prehliadači nájdeme svoju aplikáciu pod url http://localhost:3000/.

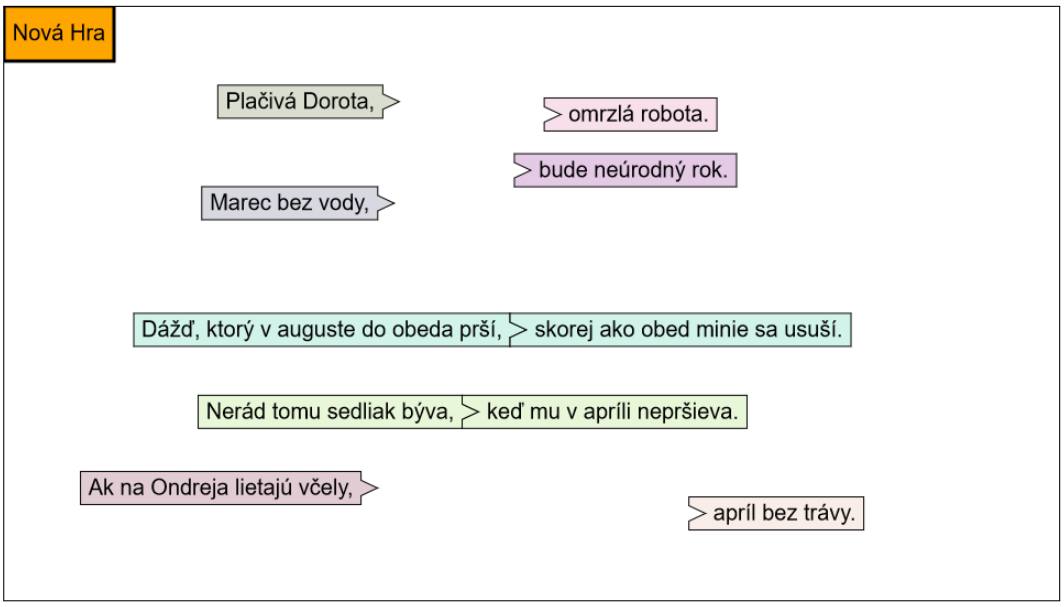
Kapitola 4

Ukážkové aplikácie

V tejto kapitole predstavíme niektoré z ukážkových aplikácií, ktoré sme vytvorili pomocou našej knižnice.

4.1 Pranostiky

Na obrázku 4.1 vidíme ukážku aplikácie „Pranostiky“. Úlohou žiaka je spojiť dve časti pranostík, ktoré sa k sebe hodia. Týmto spôsobom sa naučí ľudovú slovesnosť. Žiak spája kartičky presúvaním tak, aby do seba pasovali. Pri spojení správneho páru sa kartičky spoja a nastavia na rovnakú farbu.

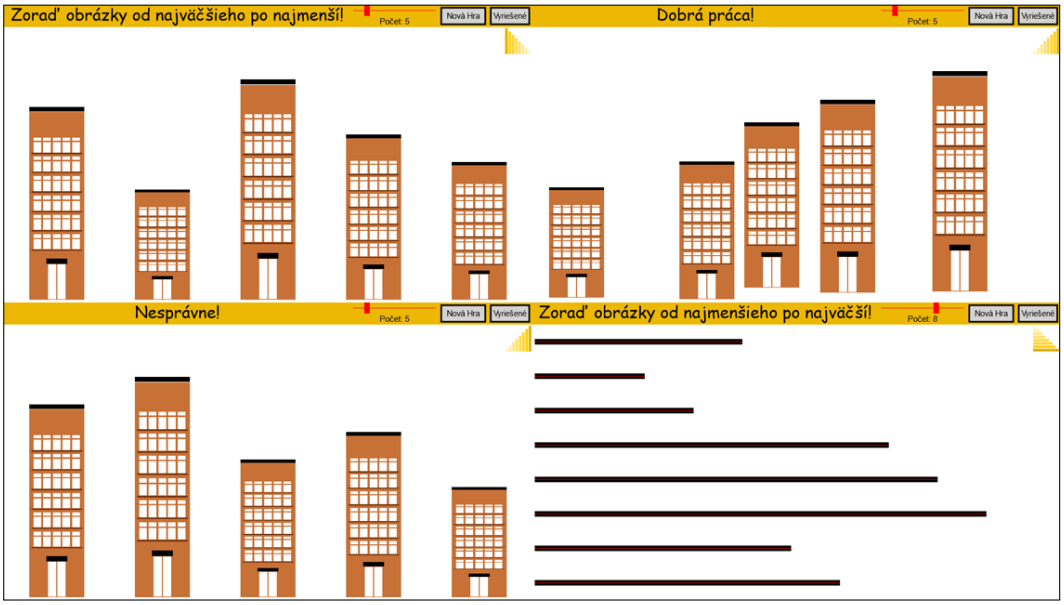


Obr. 4.1: Ukážka aplikácie Pranostiky.

Tvar kartičiek je zobrazený ako *polygon* pomocou triedy **GameShape**. To, kedy sa prekrývajú, zisťujeme pomocou hitboxov (spomínané v časti 3.2).

4.2 Zoraď obrázky

Na obrázku 4.2 vidíme ukážku aplikácie „Zoraď obrázky“. Úlohou žiaka je presúvať obrázky po plátne podľa zadania. Žiak má okrem písomného zadania aj pomôcku v pravom hornom rohu, ktorá mu ukazuje správny smer. Táto aplikácia sa dá využiť na hodine matematiky na výučbu porovnávania a zoraďovania.



Obr. 4.2: Ukážky aplikácie Zoraď obrázky. Na ukážke vidíme zadanie úlohy, správne a nesprávne riešenie. Okrem vertikálneho zoraďovania je možné aj horizontálne.

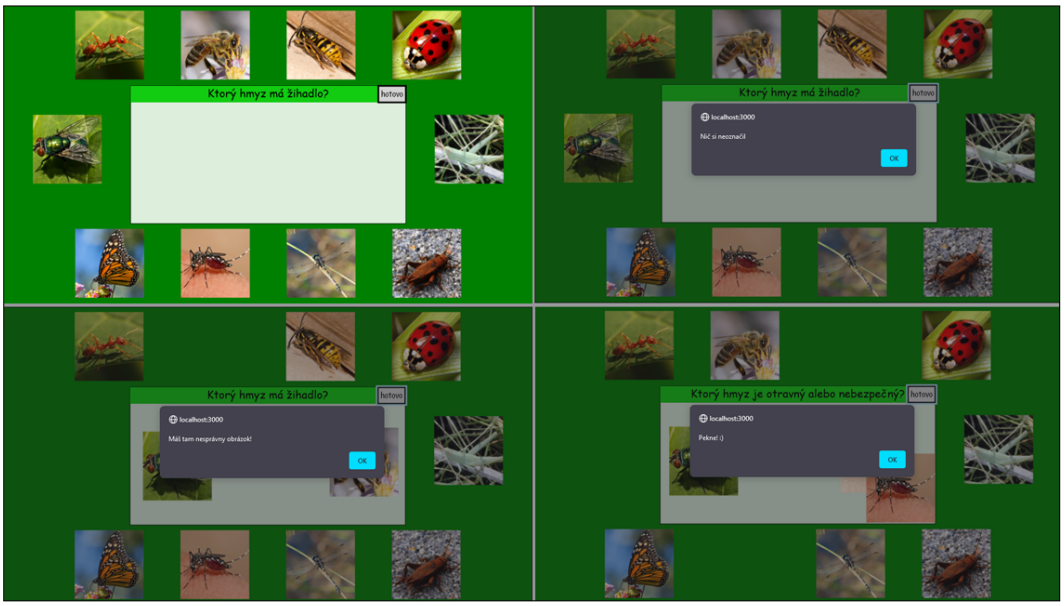
Pri kontrole správnosti porovnávame pozíciu obrázkov na horizontálnej alebo vertikálnej osi, podľa typu úlohy.

Zobrazovanie obrázkov rôznej veľkosti riešime cez atribút *height*. Pomôcka v pravom hornom rohu je obrázok, ktorého smer (zrkadlenie) meníme atribútom *hScale* a rotáciu cez *rotation*. Pomocou týchto atribútov nie je potrebné ukladať viac verzií obrázkov podľa veľkosti, lebo priamo v kóde meníme spôsob ich zobrazenia.

V aplikácii využívame aj element **GameRangeSlider**, pomocou ktorého si žiak môže nastaviť počet zobrazených objektov.

4.3 Poznáš hmyz?

Na obrázku 4.3 vidíme ukážku aplikácie „Poznáš hmyz?“. Úlohou žiaka je presunúť obrázky do vyznačeného poľa podľa zadania. Pri stlačení tlačidla „kontrola“ dostane správu o správnosti svojho riešenia.



Obr. 4.3: Ukážka aplikácie Poznáš hmyz? Na ukážke vidíme rôzne správy podľa správnosti vybraných obrázkov.

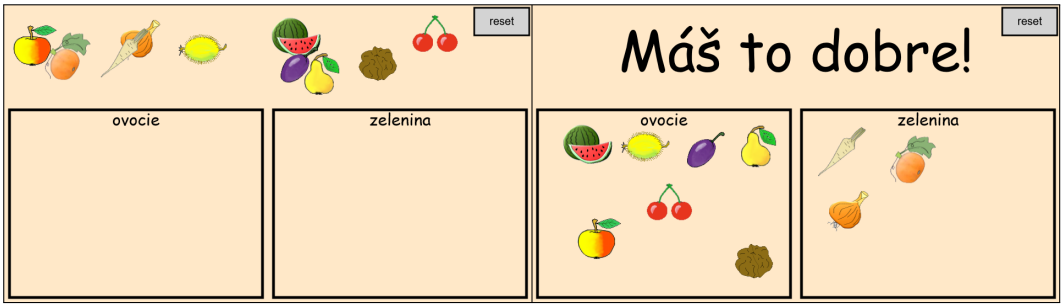
Pri kontrole správnosti využívame metódu *isInside* na elemente predstavujúcom pole označených obrázkov. Podľa toho, či sú zvolené všetky obrázky, nejaký tam chýba alebo sú medzi zvolenými nesprávne, sa pri kontrole zobrazí správa. Zobrazenie správy sme riešili pomocou funkcie `window.alert()`, ktorá je definovaná v prehliadačoch.

4.4 Vyber obrázok, ktorý tam nepatrí

Na obrázku 4.4 vidíme ukážku aplikácie „Vyber obrázok, ktorý tam nepatrí“. Úlohou žiaka je kliknúť na obrázok, ktorý spĺňa podmienku zadania. Táto aplikácia pomáha s rozvojom logického myslenia a schopnosťou zaradiť objekty do skupín podľa ich vlastností.

Obr. 4.4: Ukážka aplikácie Vyber obrázok, ktorý tam nepatrí. Na ukážke vidíme vľavo dole počítadlo bodov za každú správnu voľbu.

4.5 Rozdeľ ovocie a zeleninu



Obr. 4.5: Ukážka aplikácie Rozdeľ ovocie a zeleninu.

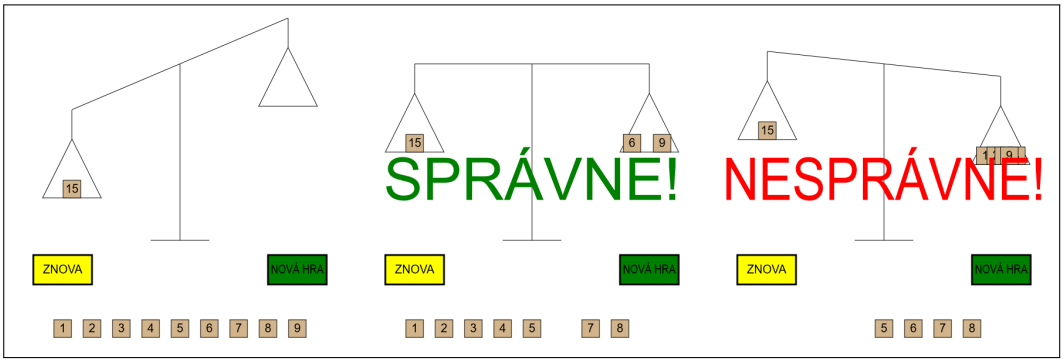
Na obrázku 4.5 vidíme ukážku aplikácie „Rozdeľ ovocie a zeleninu“. Úlohou žiaka je presunúť a rozdeliť obrázky do „krabíc“, do ktorých patria. Keď sú všetky obrázky v správnych krabiciach, zobrazí sa správa o výhre. Žiak môže kedykoľvek stlačiť tlačidlo „reset“, ktoré spustí hru odznova a zobrazí náhodný výber obrázkov.

4.6 Vyváž váhy

Na obrázku 4.6 vidíme ukážku aplikácie „Vyváž váhy“. Táto aplikácia je zjednodušená verzia aplikácie z ukážky 1.3. Úlohou žiaka je presunúť závažia na pravú stranu váh tak, aby boli vyvážené.

Uhol naklonenia váh podľa zaťaženia sme dosiahli kombináciou atribútu *rotation* a využitia elementu **GameComposite**. Ramená váh predstavujú kompozit elementov,

ktorý rotujeme o vypočítaný uhol okolo bodu na vrchu váh.



Obr. 4.6: Ukážka aplikácie Vyváž váhy.

4.7 Spoj obrázky čiarou

Obr. 4.7: Ukážka aplikácie Spoj obrázky čiarou.

Na obrázku 4.7 vidíme ukážku aplikácie „Spoj obrázky čiarou“. Úlohou žiaka je kliknúť na obrázok a ťahať z neho čiaru, ktorou má prepojiť súvisiace obrázky. Keď spojí správnu dvojicu, ťahaný obrázok sa presunie na cieľový.

Čiaru, ktorou spájame obrázky, reprezentujeme elementom s tvarom čiary (typ *line* triede **GameShape**), ktorej súradnice (začiatok a koniec) nastavujeme pri ťahaní pomocou metódy `setLine`.

Kapitola 5

Testovanie knižnice

Knižnica bola testovaná dvoma pedagógmi, ktorí v minulosti tvorili malé edukačné aplikácie pre žiakov.

Úlohou testerov bolo vytvoriť pomocou knižnice nejakú jednoduchú aplikáciu (prípadne viac) a ohodnotiť, ako jednoducho sa im s knižnicou pracovalo. Obtiažnosti s prácou s knižnicou a ich pripomienky k funkcionalite sme zapracovali do knižnice.

V tejto kapitole popíšeme funkcionalitu, ktorú sme pridali do knižnice počas procesu testovania.

5.1 Úprava funkcionality slidera

Trieda **GameRangeSlider** predstavuje slider, ktorý si uchováva nejakú hodnotu. Pôvodné riešenie bolo uchovanie si hodnoty medzi 0 a 1, čo predstavovalo percentuálnu hodnotu.

Keďže slider sa v edukačných aplikáciách využíva skoro exkluzívne na celé čísla v rozsahu, zmenili sme funkcionalitu nasledovne:

- Pridali sme atribúty *max* a *min*, ktoré reprezentujú rozsah slidera. Tiež sme pridali metódu `setBounds(min,max)`, ktorá spravuje nastavenie rozsahu.
- Pridali sme atribút *floating*, ktorý rozhoduje o tom, či slider bude vracať celočíselnú alebo desatinnú hodnotu z daného rozsahu.

5.2 Často používané funkcie

Pri tvorbe edukačných aplikácií sa okrem základných funkcií na zobrazovanie, eventov a ostatných, ktoré naša knižnica ponúka ako vlastnosti tried, často vyskytujú postupy, ktoré by nemalo byť nutné opakovať.

Pri testovaní sme odhalili tieto často používané funkcie:

- **Premiešanie prvkov poľa.** Javascript neponúka žiadnu vstavanú funkciu na premiešanie poľa. Na tieto účely sme vytvorili funkciu `shuffleArray(array)`, ktorá zamieša pole a vráti ho. Táto funkcia nemení pôvodné pole.

- **Náhodný výber niekoľkých elementov poľa.** V prípade, že máme veľké pole a chceme z neho vybrať určité množstvo náhodných prvkov, treba pole zamiešať a vybrať z neho prvých toľko prvkov, koľko potrebujeme. Na tento účel sme vytvorili funkciu `randomSelection(arr,length)`, ktorá zo vstupného poľa vytvorí pole náhodných prvkov žiadanej dĺžky.

- **Náhodná svetlá farba.** V niektorých aplikáciách chceme z estetických dôvodov priradiť objektom náhodnú farbu. Táto farba ale nemôže byť príliš tmavá, aby to nevytváralo problém s čitateľnosťou textu. Pre tento prípad sme vytvorili funkciu `randomLightColor()`, ktorá vráti takúto farbu v hexadecimálnom tvare. Knižnica ponúka aj funkciu `randomColor()`, ktorá vráti náhodnú farbu, ktorá ale už nemusí byť svetlá.

- **Zmazanie objektu z poľa.** Na vymazanie referencie z poľa treba použitím JavaScriptu využiť kombináciu funkcii `indexOf` a `splice`. Využitím tohto spôsobu sa vymaže iba prvá referencia v poli, v prípade potreby viacnásobného mazania je potrebné zisťovať index viackrát. Kvôli tomu sme vytvorili funkciu `removeFromArray(array,element,removeAll)`, ktorá zastupuje túto funkcionalitu. Užívateľ si môže nastaviť, či chce zmazať iba prvú referenciu, alebo nastavením parametra *removeAll* vymaže z poľa všetky referencie na objekt.

- **Náhodné celé číslo z rozsahu.** JavaScript ponúka funkciu `random()`, ktorá vráti náhodné desatinné číslo v rozsahu medzi 0 a 1. Číslo je ďalej potrebné násobiť žiadaným rozsahom, zaokrúhliť nadol a posunúť o (pričítať k nemu) spodnú hranicu na to, aby z neho bolo celé číslo v rozsahu. Tento proces sa často opakuje, a preto je na mieste ho zjednodušiť. Do knižnice sme pridali funkciu `randomInt(min,max)`, ktorá vráti celé číslo v rozsahu medzi *min* a *max*.

5.3 Jednoduchosť zmeny pozície elementov

Pozícia elementov sa dá nastaviť a meniť niekoľkými spôsobmi, spomínané v časti 3.1.2. Pôvodne, v prípade potreby zmeny iba jednej súradnice, musel programátor pristupovať k atribútu *center* a meniť jeho vlastnosti cez `element.center.x = ...`, alebo si niekde uchovať druhú súradnicu a využiť metódu `setPosition(x,y)`.

Pre tieto účely sme vytvorili gettery a settery pre súradnice elementu, aby k nim programátor vedel pristupovať a meniť ich priamo pomocou `element.x = ...`.

5.4 Inicializácia elementu s pozíciou

Pri inicializácii elementu pomocou metódy `game.createElement()` nebolo možné nastaviť pozíciu. Pôvodne sme to riešili tak, že za inicializáciou nasledovala funkcia `setPosition`, ktorá element umiestnila.

Preto sme pridali možnosť inicializovať element s atribútmi x a y, ktoré nastavia elementu pozíciu.

5.5 Domovská pozícia elementu

V edukačných aplikáciách sa často vyskytuje to, že element sa pri presunutí na nesprávne miesto vráti na pôvodnú pozíciu. Kvôli tomu si tak programátor musí ukladať pozíciu elementu a presúvať ho na pôvodné miesto.

Na zjednodušenie tohto procesu sme vytvorili funkciu `home()`, ktorá vráti element na domovskú pozíciu, uloženú v triednej premennej *homePosition*. Táto pozícia je automaticky nastavená pri inicializácii elementu pomocou atribútov x a y, alebo sa dá kedykoľvek zmeniť pomocou funkcie `setHome(x,y)`.

5.6 Presunutie ťahaného elementu na vrch

Ťahaný element je často potrebné presunúť na vrch, pre jednoduchosť ďalšej práce s ním. Kvôli tomu sme elementu pridali atribút *keepOnTop*, ktorý zabezpečí, že presúvaný element ostane na vrchu všetkých zobrazených.

Na to, aby sa presúvaný element nezobrazil nad tlačidlami, alebo elementami, ktoré majú nastavený atribút *level* na `Number.POSITIVE_INFINITY`, čo predstavuje najvyššiu úroveň, sme oddelili zobrazovanie elementov typu **GameButton**, **GameRangeSlider** a **GameTextInput** od ostatných tak, aby boli vždy na vrchu. Takto atribút *level*, ktorý je týmto objektom priradený, hovorí len o ich úrovni medzi sebou a nie je potrebné ho nastaviť na nekonečno. Úroveň presúvaných elementov sa ďalej neporovnáva s úrovňami nastavenými ako nekonečno.

5.7 Klikanie do herného poľa cez event `onMouseDown`

Pri pridávaní elementov cez klikanie do herného poľa nastal problém, že pri každom kliku sa pridal ďalší. Takto sa nedalo manipulovať so zobrazenými elementami, ale iba s novo pridanými.

Tento problém sme vyriešili tak, že event onMouseDown sa spustí, iba ak užívateľ klikne na prázdne miesto a nie na element.

5.8 Definícia mnohouholníkov či čiar ako „korytnačka“

Mnohouholníky je potrebné definovať súradnicami. Na zistenie týchto súradníc je často potrebné ich vypočítať na kalkulačke, alebo napísať si pomocný skript na získanie žiadaných hodnôt.

Zložité počítanie súradníc sa dá čiastočne zredukovať pomocou „krokovania“. Užívateľ knižnice môže okrem využitia pevných súradníc napísať postup v jazyku korytnačky, akým sa bude tvar kresliť. Do inicializácie tvaru (**GameShape**) sme pridali atribút *path*, pomocou ktorého program vytvorí súradnice pre tvar. Princíp je nasledovný:

- Cesta je reprezentovaná ako `string`,
- Na začiatku užívateľ nastaví štartovaciu pozíciu relatívne k elementu, napríklad `"0 0"`,
- Ďalej môžu nasledovať príkazy *f - forward, b - backward*, za ktorými užívateľ zadá vzdialenosť v pixeloch,
- Tiež môže využiť príkazy *r - right, l - left*, za ktorými nasleduje uhol v stupňoch.

Keby užívateľ chcel týmto spôsobom nakresliť štvorec, mohol by to spraviť ako `"-50 -50 f 100 r 90 f 100 r 90 f 100 r 90 f 100"`. Pre opakované príkazy tiež môže využiť funkciu `String.repeat()` ako `"-50 -50" + " f 100 r 90".repeat(4)`.

Okrem definovania cesty pri inicializácii môže užívateľ kedykoľvek nastaviť cestu tvaru pomocou funkcie `setPath(path)`.

5.9 Názor testerov na prácu s knižnicou

Našou úlohou pri tvorbe knižnice bola okrem prepracovanej funkcionality aj pohodlnosť jej využívania. V dotazníku k záveru testovania sme sa pýtali, ako sa testerom pracovalo s knižnicou. Dozvedeli sme sa tieto body:

- Inštalácia knižnice pomocou npm bola zvládnuteľná a návod bol postačujúci.
- Testerom sa s knižnicou pracovalo veľmi dobre a zvládli spraviť všetko, čo chceli.
- Testeri nemali žiadne námietky k existujúcej štruktúre knižnice a nemali v nej potrebu nič meniť.

Pri procese testovania sme objavili niekoľko možných vylepšení knižnice, ktorým sme sa v tejto kapitole venovali. Skoro všetka funkcionalita sa však dala dosiahnuť aj bez žiadaných zmien.

Kapitola 6

Možné rozšírenia knižnice

Pri tvorbe knižnice sme zaobalili určitú funkcionalitu, ktorá sa často využíva pri tvorbe edukačných hier. Táto knižnica však so sebou nesie určité limitácie, ktoré sa neskôr dajú vyriešiť.

V tejto kapitole uvedieme možné zmeny a rozšírenia knižnice, ktoré by vedeli zjednodušiť, alebo zlepšiť prácu s knižnicou.

6.1 Nastavenie priehľadnosti pre farby

Pri tvorbe aplikácií pomocou našej knižnice môže programátor priradiť objektom rôzne farby. Ak sa tieto elementy prekrývajú, element na spodnej časti nie je vidno. S momentálnou funkcionalitou sa to dá obísť využitím čiastočne transparentného obrázka.

V budúcnosti by však mohla byť do knižnice pridaná možnosť nastaviť objektom čiastočnú transparenciu.

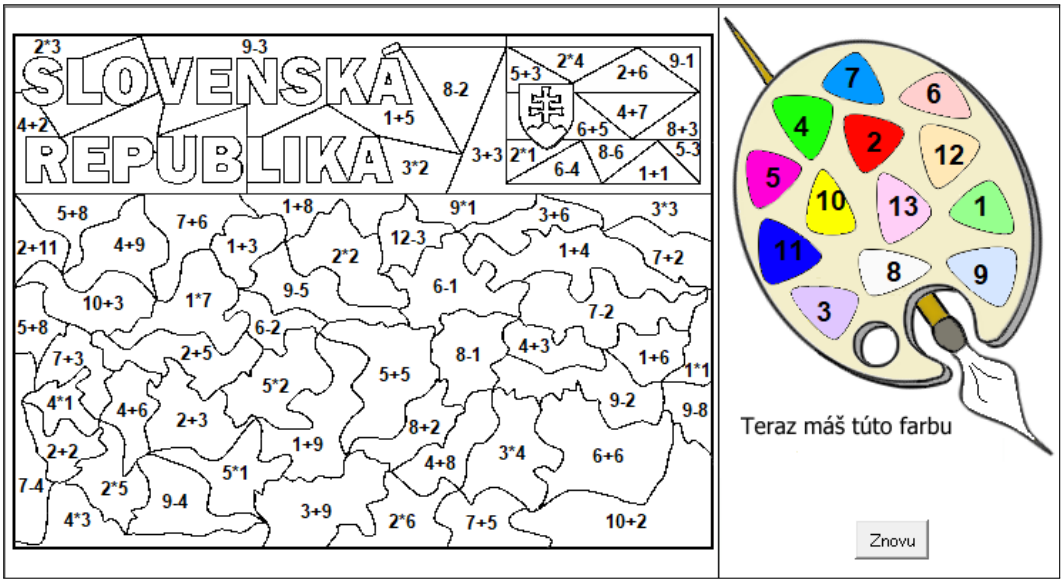
6.2 Optimalizácia rýchlosti vykresľovania objektov

Naša knižnica pri každom prekreslení plátna volá vykresľovacie funkcie pre každý drawable objekt. Tento proces sa dá urýchliť pomocou ukladania vykreslených tvarov do dočasnej pamäti ako obrázky. Takto by nebolo potrebné pri vykresľovaní vypočítavať pozíciu, ani čítať atribúty, stačilo by vykresliť obrázok uložený ako mapa bitov.

Možnosti optimalizácie pre tento proces sú viaceré. Bolo by potrebné pridať obojsmernú referenciu pre drawables a elementy, aby sa pri zmene drawables prepísal uložený obrázok v elemente. Alternatívne by tento spôsob mohla využiť iba trieda **GameShape**, ktorá má veľa nastaviteľných atribútov a proces vykresľovania vie byť pomalší pre veľké množstvo súradníc, napríklad pre typ *polygon*.

6.3 Implementácia metódy na flood fill

Edukačné aplikácie vedia byť stavané ako interaktívna omaľovánka. Na obrázku 6.1 vidíme takúto aplikáciu.



Obr. 6.1: Ukážka aplikácie Omaľovánka.

Úlohou žiaka je vymaľovať obrázok podľa zadania. V prípade ukážky 6.1 majú jednotlivé farby priradené čísla, regióny na obrázku sú značené príkladom. Žiak vypočíta príklad a vymaľuje región podľa výsledku.

Využitím funkcionality našej knižnice by sme mohli zvoliť jeden z dvoch spôsobov na riešenie tohto problému:

1. **Využitie triedy GameShape.** Tento spôsob by bol vhodný iba pri jednoduchých tvaroch. Užívateľ by musel zadať pevné súradnice objektov, ktorých farba by sa menila pri kliku.

2. **Využitie triedy GameImage.** Užívateľ si uloží viaceré verzie obrázkov, ktoré reprezentujú vyfarbiteľný región. Pri kliku na obrázok sa načíta iný, podľa zvolenej farby. Tento spôsob si ale vyžaduje buď umiestňovanie obrázkov na pevné súradnice, alebo vytvorenie obrázkov veľkosti plátna, kde je ovplyvňovaný región vopred umiestnený na správnom mieste.

Keby sme do knižnice pridali metódu, ktorá vie zmeniť pixely obrázka na zadanú farbu, vedeli by sme tento problém vyriešiť využitím obrázka, ktorý má regióny oddelené čiarami. Užívateľ by mohol zavolať metódu, ktorá na určitých súradniciach rekurzívne mení farbu pixelov, dokým nenarazí na hranicu regiónu, znázornenú inou farbou.

Správnosť riešenia by sa potom mohla overiť pomocou kontroly farby pixela na súradniciach, ktoré si už užívateľ definuje.

6.4 Multiplayer

V edukačných aplikáciách sa niekedy môže využiť aj komunikácia medzi počítačom a serverom. Napríklad by sme mohli sledovať, ktorý žiak vyrieši úlohu ako prvý a zobraziť tabuľku podľa času. Tiež môžme implementovať hru pre viac hráčov, kde súťažia, alebo spolupracujú.

Naša knižnica túto funkcionalitu neponúka, preto ak by ju chcel programátor zakomponovať, musel by využiť *WebSocket-y* ponúkané javascriptom, alebo alternatívy vo forme modulov pre Node.js.

Keby sme túto funkcionalitu chceli pridať do knižnice, mohli by sme vytvoriť triedu server, ktorá by spracúvala rôzne inštancie hier (aplikácií) a riadila by komunikáciu medzi počítačmi.

Komunikácia by bola vedená cez posielanie textových príkazov medzi počítačom a serverom, v našich triedach by museli byť pridané metódy na spracovanie týchto príkazov.

V prípade implementácie tejto funkcionality bude treba nejako vyriešiť to, ako sa bude kontrolovať synchronizácia objektov, ako sa budú riešiť prípadné konflikty s časovaním (ako napríklad zachovanie poradia príkazov od viacerých počítačov serverom) a ďalšie.

6.5 GUI pre tvorbu aplikácií

Pri tvorbe aplikácií pomocou našej knižnice sa využíva umiestňovanie na súradnice a nastavovanie veľkosti, rotácie, farby a podobne v kóde. Keby naša knižnica ponúkala spôsob, ako sa dajú objekty umiestniť na plátno ručne (bez zadávania súradníc v kóde), tento proces by sa zjednodušil.

Najjednoduchší spôsob, ako sa toto dá implementovať, je využitím HTML elementu `<form>` a prvkami `<input>`, ktorými sa budú nastavovať vlastnosti objektov. Užívateľ by mal na obrazovke zobrazené plátno, ktoré predstavuje výsledný produkt, dynamické

elementy `<input>`, ktoré spracúvajú nastavenie zvoleného elementu a textové pole s vygenerovaným kódom, ktorý môže vložiť do svojej aplikácie.

Takto by sa mohlo spracúvať nahrávanie obrázkov, umiestňovanie elementov, nastavovanie vlastností objektov, meno premenných a ďalšie. Užívateľ potom môže pre vygenerovaný kód písať funkcie pre interakcie medzi elementami. Prípadne by sa mohlo využiť prostredie podobné jazyku *scratch* na definovanie funkcionality priamo vizuálne.

6.6 Ďalšie špecifické elementy

Naša knižnica poskytuje určité často používané elementy, ako napríklad tlačidlá či slider. Avšak stále sa tam dajú pridať ďalšie, ktoré naša knižnica neponúka, ako napríklad checkbox alebo radiobutton.

Užívateľ, s použitím našej knižnice, ich vie vytvoriť ako element, ktorému zadefinuje eventy na klikanie, avšak tento proces je zložitejší ako využitie preddefinovaných elementov.

Knižnicu je možné rozšíriť o nové elementy.

6.7 Využitie jazyku TypeScript

Naša knižnica je písaná v jazyku JavaScript. JavaScript je client-side skriptovací jazyk, ktorý podporuje využitie tried. TypeScript je objektovo-orientovaný jazyk, pre ktorý triedy sú jedným zo základných blokov. Keby bola naša knižnica písaná v TypeScript-e, zjednoduší sa proces odchytávania chýb, pretože sa ukážu už počas kompilácie.

Keďže JavaScript je loosly-typed (premenné nemusia mať konkrétny typ), narážame na problém pri definovaní funkcií, keďže prostredie, v ktorom programujeme (IDE), nevie, akého typu má byť parameter, alebo aký typ vráti funkcia.

Túto limitáciu obchádzame v našej knižnici využitím Node.js modulu JsDoc, pomocou ktorého vytvárame dokumentáciu kódu, ktorú vie čítať aj IDE.

6.8 Spätná kompatibilita

Pre potreby našej knižnice sme sa sústredili na jej kompatibilitu s modernými prehliadačmi na rôznych zariadeniach. Avšak aj v súčasnej dobe využíva menej ako 1% užívateľov zastaralé prehliadače ako Internet Explorer (IE).

IE nepodporuje využitie modulov, ani moderné vlastnosti JavaScript-u (ES6), ako napríklad arrow functions a ďalšie. Písať kód fungujúci na takomto prehliadači je zbytočne namáhavé a spomaľuje to dobu vývoja.

Pre účely spätnej kompatibility existujú takzvané transpilátory, ako napríklad *babel*, ktoré konvertujú kód písaný v štandarde ES6 na staršie verzie, ktoré sú funkčné na IE.

V budúcnosti by mohla byť naša knižnica kompatibilná aj pre staršie zariadenia cez využitie transpilátora.

Záver

Po analýze pracovných zošitov, existujúcich jednoduchých edukačných aplikácií a existujúcej bakalárskej práce sme navrhli štruktúru knižnice, ktorá dáva k dispozícii všetky nástroje, ktoré užívateľ môže pri tvorbe aplikácií využiť. Navrhnutú štruktúru sme ďalej implementovali, vytvorili sme pomocou nej niekoľko ukážkových aplikácií a dali sme ju otestovať pedagógom, ktorí majú s tvorbou edukačných aplikácií skúsenosti. Knižnica je plne funkčná a ponúka jednoduchý spôsob, ako vytvárať aplikácie s často opakovanými konceptami, ale dajú sa pomocou nej vytvárať aj zložitejšie aplikácie či hry, ktoré nemusia byť len edukačné.

Vo východiskovej kapitole sme analyzovali pracovné zošity a niekoľko edukačných aplikácií pre cieľovú vekovú kategóriu žiakov, klasifikovali sme úlohy v nich do niekoľkých skupín a popísali, aké typy objektov a aké typy manipulácie s týmito objektami sa v nich najčastejšie vyskytujú. V tejto kapitole sme tiež popísali technológie zvolené na vytvorenie knižnice. Rozhodli sme sa pre implementáciu knižnice využiť HTML element `<canvas>` a čistý JavaScript. Pre zjednodušenie inštalácie sme využili Node.js, k nemu pribalený manažér balíkov *npm* a modul Express. Pre obídenie limitácií JavaScript-u a vytváranie dokumentácie sme využili aj modul JSDoc.

Na základe záverov z východiskovej kapitoly sme vytvorili návrh knižnice, ktorý sme ďalej implementovali. V kapitole Implementácia sme sa venovali spôsobom, ktoré sme pri tvorbe knižnice zvolili. Analyzovali sme v nej aj rôzne spôsoby, ktoré sme sa rozhodli nevyužiť a k tým, ktoré sme zvolili, sme pridali aj názorný kód, ktorý sme v knižnici využili.

Výsledkom implementácie je plne funkčná knižnica, pomocou ktorej sme vytvorili súbor ukážkových aplikácií, ktoré sme predstavili v kapitole Ukážkové aplikácie. Pri predstavovaní aplikácie sme ukázali, ako sme implementovali niektorú funkcionalitu pomocou našej knižnice.

V kapitole Testovanie sme uviedli niekoľko zmien, ktoré sme na knižnici vykonali počas procesu testovania. Odhalili sme niektoré často používané funkcie, skrátili sme niektoré často používané procesy, ako napríklad presúvanie objektov a tiež sme na základe pripomienok od testerov pridali funkcionalitu, ktorá zvyšuje hodnotu knižnice.

Na záver sme predstavili možné rozšírenia výslednej knižnice. Knižnica je plne funkčná a v mnohých veciach zjednodušuje tvorbu edukačných aplikácií, ale chýba v

nej funkcionalita napríklad na prepojenie viacerých zariadení na sieti, ktorú by užívateľ pri tvorbe aplikácií mohol využiť.

Táto knižnica je k dispozícii na stiahnutie pomocou *npm* a jej kód je dostupný a voľne šíriteľný online. Pri tvorbe knižnice sme kládli dôraz na jednoduchosť jej využívania, inštalácie a prípadného rozširovania.

Na záver testovania nám testeri vyplnili dotazník o spokojnosti s knižnicou a dostali sme len pozitívne ohlasy. Práca s knižnicou je jednoduchá a podobá sa na v minulosti používané zastaralé pluginy, ktoré testeri využívali. Knižnica okrem zjednodušenia práce s často opakovanými konceptami ponúka štruktúru, pomocou ktorej je možné vytvoriť aj zložitejšie aplikácie. Ciele práce preto považujeme za naplnené.

Literatúra

[1] TITKOVÁ R., SIPOSOVÁ E., BULEJOVÁ T., KULICHOVÁ A., 2021, *Slovenský jazyk pre 2. ročník ZŠ, 2. časť.* Aitec spol. s.r.o. 2021. ISBN 978-80-8146-202-3

[2] BELIC M. – STRIEŽOVSKÁ J., 2019, *Matematika pre štvrtákov – pracovný zošit 2. časť.* Aitec spol. s.r.o. 2019. ISBN 978-80-8146-162-0

[3] INFOVEKÁČIK. 2003-2013, *Internetový časopis Infovekáčik.* [online]. Asociácia projektu Infovek, 2016. [cit. 15.1.2022] Dostupné na internete: `http://infovekacik.edu.fmph.uniba.sk/`

[4] VIKI. 2019, *Centrálne úložisko digitálneho edukačného obsahu.* [online]. Ministerstvo školstva, vedy, výskumu a športu Slovenskej republiky 2019. [cit. 15.1.2022] Dostupné na internete: `https://viki.iedu.sk/`

[5] BOHUNICKÁ I., 2018, *Pythonovský framework pre vytváranie hier*: bakalárska práca. Univerzita Komenského v Bratislave FMFI FMFI.KAI. Bratislava, 2018.

[6] NODE.JS, 2022, *Oficiálna dokumentácia systému Node.js.* [online]. OpenJS Foundation, 2022. [cit. 15.1.2022] Dostupné na internete: `https://nodejs.org/en/docs/`

[7] EXPRESS, 2017, *Oficiálna dokumentácia balíka Express pre Node.js.* [online]. OpenJS Foundation, 2017. [cit. 15.1.2022] Dostupné na internete: `http://expressjs.com/en/4x/api.html`

[8] JSDOC 3, 2017, *Oficiálna dokumentácia balíka JSDoc pre Node.js.* [online]. Open Source Project, 2011-2017. [cit. 10.3.2022] Dostupné na internete: `https://jsdoc.app/`

[9] TASR, 2014, *Tablety sa v školách osvedčili, chcú ich učitelia i žiaci.* [online]. Tlačová agentúra Slovenskej Republiky, 2014. [cit. 25.4.2022] Dostupné na internete: `https://www.minedu.sk/tablety-sa-v-skolach-osvedcili-chcu-ich-ucitelia-i-ziaci/`

Prílohy

Priložený zip obsahuje:

1 Zdrojový kód knižnice, tiež dostupný na stránke `https://github.com/hutnikus/easy-educational-games`.

2 Zdrojový kód vzorového projektu, tiež dostupný na stránke `https://github.com/hutnikus/easy-educational-games-testing`.

3 Python script pre formátovanie animovaných GIFov.

4 Podrobný triedny diagram.