

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Robomise v mobilnom prostredí

BAKALÁRSKA PRÁCA

Miriama Zemaníková

Brno, jar 2019

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Robomise v mobilnom prostredí

BAKALÁRSKA PRÁCA

Miriama Zemaníková

Brno, jar 2019

Na tomto mieste sa v tlačenej práci nachádza oficiálne podpísané zadanie práce a vyhlásenie autora školského diela.

Vyhlásenie

Vyhlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracovala samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používala alebo z nich čerpala, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Miriama Zemaníková

Vedúci práce: Mgr. Marek Grác, Ph.D.

Podakovanie

Na tomto mieste by som sa rada poďakovala vedúcemu tejto diplomovej práce, Mgr. Marekovi Grácovi, Ph.D., za odborné rady, cenné pripomienky a čas, ktorý tejto diplomovej práci venoval.

Zhrnutie

Bakalárska práca sa zaoberá implementáciou mobilnej aplikácie Robomise, ktorá vychádza z požiadaviek užívateľov, ktorí otestovali jej webový variant. Táto natívna aplikácia slúži na rozvoj algoritmického myslenia a má formu hry.

Kľúčové slová

android, mobilná aplikácia, react native

Obsah

1	Úvod	1
1.1	<i>Motivácia</i>	1
1.2	<i>Ciele práce</i>	2
1.3	<i>Štruktúra práce</i>	2
2	Používateľské testovanie webovej aplikácie	3
2.1	<i>Webová aplikácia Robomise</i>	3
2.1.1	Princíp hry	3
2.1.2	Hracia plocha	4
2.1.3	Úlohy a program	4
2.2	<i>Príprava testovania použiteľnosti</i>	4
2.2.1	Testovací scenár	5
2.2.2	Testovací účastníci	6
2.3	<i>Priebeh testovania</i>	6
2.4	<i>Vyhodnotenie testovania</i>	7
2.4.1	Vyhodnotenie dát	7
2.4.2	Návrhy riešení	9
3	Mobilná aplikácia	13
3.1	<i>React a React Native</i>	13
3.2	<i>Základná kostra projektu</i>	14
3.3	<i>Navigácia</i>	15
3.4	<i>Správa stavov</i>	16
3.5	<i>Api</i>	17
3.6	<i>Štýly</i>	19
3.7	<i>Komponenty hry Robomise</i>	21
3.7.1	TaskTable	21
3.7.2	SpaceGame	22
3.7.3	BlocklyEditor	24
3.8	<i>Postrehy z vývoja</i>	31
4	Používateľské testovanie mobilnej aplikácie	33
4.1	<i>Vyhodnotenie dát</i>	33
4.2	<i>Návrhy riešení</i>	34
5	Záver	37

A	Príloha	39
A.1	<i>Testovací scenár</i>	39
A.1.1	Zázemie účastníkov	39
A.1.2	Testovacie zázemie	39
A.1.3	Pred testovaním	39
A.1.4	Uzavretie testovania	40
A.1.5	Ciele testovania	40
A.1.6	Testovacie otázky	40
A.2	<i>Záznam webového používateľského testovania</i>	40
A.2.1	Zázemie účastníkov	40
A.2.2	Testovacie zázemie	41
A.2.3	Uzavretie testovania	42
A.3	<i>Záznam mobilného používateľského testovania</i>	43
A.3.1	Zázemie účastníkov	43
A.3.2	Testovacie zázemie	43
A.3.3	Uzavretie testovania	44
	Bibliografia	45

Zoznam tabuliek

Zoznam obrázkov

2.1	Príklad riešenia úlohy hry Robomise	5
2.2	Úloha Obklíčený diamant	8
2.3	Vylepšená úloha Obklúčený diamant	10
3.1	Schéma architektúry React Native a porovnanie s ReactJS.	14
3.2	Webová komponenta TaskTable	21
3.3	Mobilná komponenta TaskTable	22
3.4	Výsledné prostredie úlohy mobilnej aplikácie	30
3.5	Príklad málo hovoriacej chybovej hlášky v React-Native	32

1 Úvod

Od roku 2000 narástol počet užívateľov mobilných zariadení až päťnásobne. Podľa štúdie Cisco Visual Networking Index[1] sa má toto množstvo do roku 2020 zvýšiť až na 5,5 miliardy, čo je 70 % svetovej populácie.

Podľa štatistických dát analytickej služby StatCounter[2] sa od roku 2016 viac pristupuje na webové stránky z mobilných zariadení ako z klasických počítačov. Aktuálny percentuálny pomer prístupu z mobilných zariadení je 53,8 %.

Z prognózy Cisca a podľa štatistík StatCounter vyplýva, že je nutné vhodným spôsobom optimalizovať webové stránky, aby ich zobrazenie na mobilnom zariadení nebolo ovplyvnené kvalitou pripojenia, ako je to u webových stránok, a aby neboli ovládané iba počítačovou myšou, ktorú si k telefónu nepripojíme. Naopak, je vhodné využiť výhody mobilných zariadení, ako napríklad možnosť pracovať v offline režime alebo použitie dotykov na ovládanie celej aplikácie.

Práve z tohto dôvodu je moja práca zameraná na vývoj novej natívnej mobilnej aplikácie Robomise, ktorej predlohou je webová aplikácia rovnakého názvu.

1.1 Motivácia

Mobilná aplikácia pre zariadenie pracujúce na mobilnej platforme Google Android, vytvorená v rámci tejto bakalárskej práce, je určená na rozvoj algoritmického myslenia. Cieľovou skupinou môžu byť deti, ktoré sa chcú naučiť programovať, ale aj samotní programátori, ktorých touto formou zamestnávateľ skúša ich algoritmické znalosti na pracovnom pohovore. Zameraním sa na prvú cieľovú skupinu má aplikácia formu hry. Sú v nej využívané adaptabilné algoritmy. Deti sa vďaka nim učia rýchlejšie ako pri hrách, kde sú ďalšie levely generované automaticky [3].

Webovú aplikáciu vyvinul tím Adaptabilného učenia na Fakulte informatiky Masarykovej univerzity. Hlavným účelom mobilnej aplikácie Robomise je vytvorenie nového používateľského rozhrania, ktoré je prispôbené pre dotykové ovládanie. Sekundárnym účelom mobil-

1. Úvod

nej aplikácie je testovanie použiteľnosti webovej a mobilnej aplikácie Robomise.

1.2 Ciele práce

Cieľom tejto práce je používateľským testovaním zistiť slabé a silné stránky predlohovej webovej aplikácie Robomise a na ich základe navrhnúť novú natívnu mobilnú aplikáciu. Tento cieľ práce je rozdelený do niekoľkých menších cieľov.

- a) Prvým cieľom je, na základe analýzy a testovania používateľskej štruktúry celého systému súčasnej webovej aplikácie, zistiť jej nedostatky a navrhnúť vylepšenia.
- b) Druhým cieľom je pomocou všetkých získaných poznatkov naprogramovať novú mobilnú aplikáciu na platforme Google Android. Táto mobilná aplikácia bude čiastočne zastupovať súčasnú webovú aplikáciu. Aplikáciu budú používať hlavne používatelia, ktorí si chcú otestovať znalosti programovania alebo sa naučiť základné algoritmy. Prispôbená bude hlavne pre tablety s dostatočne veľkým displejom.
- c) Tretím cieľom je používateľsky otestovať natívnu aplikáciu a navrhnúť jej prípadné vylepšenia.

1.3 Štruktúra práce

Text práce je rozdelený do niekoľkých kapitol, ktoré odpovedajú menším cieľom. V druhej kapitole je popísaná súčasná webová aplikácia a používateľské testovanie na nej. Kľúčovou súčasťou druhej kapitoly je zoznam požiadaviek na novú aplikáciu. V tretej kapitole je popísaný návrh a implementácia novej mobilnej aplikácie. V záverečnej kapitole je pozornosť venovaná používateľskému testovaniu natívnej aplikácie a je tu uvedený zoznam možných budúcich rozšírení.

2 Používateľské testovanie webovej aplikácie

Je veľmi dôležité odstrániť problémy, ktoré by mohli používateľa frustrovať alebo donútiť ho opustiť aplikáciu. Používateľské testovanie je metóda, ktorá pomáha odhaliť, ako užívatelia používajú produkt. Vďaka tomu je možné odhaliť nedostatky a problémy aplikácie. K týmto nedostatkom dochádza kvôli tomu, že vývojari, dizajnéri a ostatní ľudia, ktorí sú spoluautormi produktu, nie sú zároveň aj jeho používateľmi.

Princípy používateľského testovania, jeho prípravu a realizáciu, ktoré boli použité v tejto bakalárskej práci, opisuje diplomová práca Mgr. Evy Vaškovej [4].

Steve Krug prirovnáva používateľské testovanie k návšteve cudzincov vo vašom meste. Mnohokrát sa totiž pri prevádzkaní používateľa aplikáciou začnú ukazovať veci, ktoré si predtým tvorcovia neuvedomili. Taktiež testovanie pomáha pochopiť, že niektoré veci nemusia byť novému používateľovi zrejmé, napriek tomu, že nám pripadajú úplne prirodzené.[5]

2.1 Webová aplikácia Robomise

Pred vytvorením prvého testovacieho scenára sa s produktom musí osoba, ktorá ho zostavuje, vopred zoznámiť a pochopiť jeho funkcionálnosť. Táto podkapitola predstavuje rozhranie testovaného softvéru a predstavuje základné pojmy, s ktorými aplikácia pracuje. Webovú aplikáciu Robomise¹ naprogramoval tím Adaptabilného učenia na Fakulte informatiky Masarykovej univerzity. Webová aplikácia Robomise je naprogramovaná v reacte. Implementačné detaily si môžete naštudovať v diplomovej práci Mgr. Tomáša Effenbergera[6].

2.1.1 Princíp hry

Webová aplikácia Robomise je hra, ktorej cieľom je hravým spôsobom naučiť používateľa algoritmicky rozmýšľať. Študent sa učí programovať riešením úloh. Preto sa pri používateľskom testovaní tester zameria hlavne na časť aplikácie, v ktorej sa úloha rieši.

1. <https://robomise.cz>

Hra má tému vesmíru a úlohou hráča je dostať vesmírnu loď do cieľovej rovinky. Počas letu nemôže vesmírna raketa naraziť do asteroidu a musí pozbierať všetky diamanty v hracom poli. Aby to nebolo úplne jednoduché a aby sa hráči naučili aj zložitejšie príkazy, ako sú napr. cykly a podmienky, sú niektoré úlohy obmedzené počtom príkazov, ktoré môžu použiť.

Používateľ sa tak vďaka hre naučí všetky základné programovacie koncepty: postupnosť príkazov, podmienky, cykly. Pri spustení programu tlačidlom Spustiť hráč vidí, ako sa raketka pohybuje a ktorý príkaz to spôsobuje, a tak v prípade, že sa mu nepodarí do cieľa doletieť úspešne, používateľ vie, na ktorý príkaz sa má sústrediť pri oprave.

2.1.2 Hracia plocha

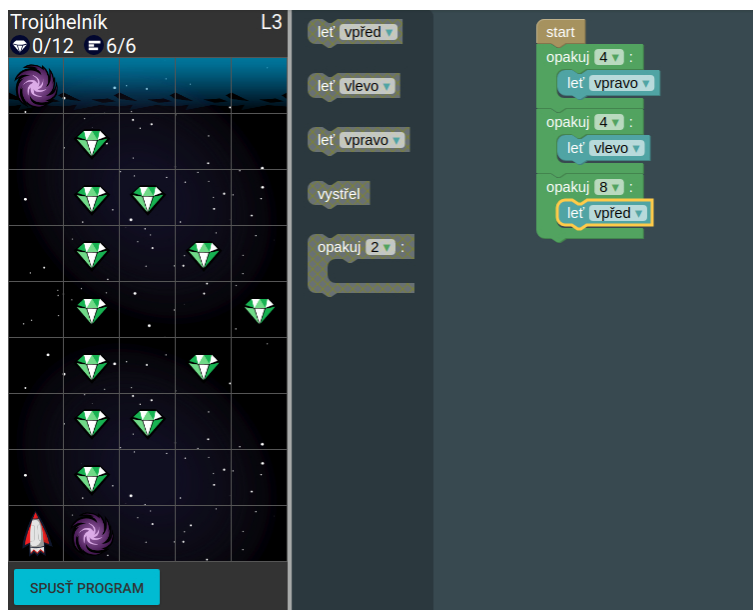
Hracia plocha, v ktorej sa vesmírna loď pohybuje, je mriežka, kde každé pole má svoju farbu, na základe ktorej sa môže raketka rozhodnúť zmeniť smer. Každé pole môže mať okrem farby aj hrací objekt. Ako napríklad diamant, ktorý musí raketka pozbierať, asteroid, ktorému sa musí vyhnúť, meteorid, ktorý môže zničiť výstrelom, a červiu dieru, ktorá slúži ako teleport. Raketka začína v dolnom riadku mriežky a každým príkazom sa posunie o jeden riadok nahor.

2.1.3 Úlohy a program

Každá úloha je riešená skladaním blokov príkazov, ako môžeme vidieť na obrázku 2.1. Tieto bloky presúva používateľ pomocou myši z kontajnera dostupných blokov a pripája ich za blok Štart. Náročnosť úloh sa postupne zvyšuje a s narastajúcou úrovňou pribúdajú aj nové bloky, reprezentujúce príkazy programu. Hra obsahuje viac ako 80 úloh, rozdelených do 9 levelov.

2.2 Príprava testovania použiteľnosti

Príprava a plánovanie testovania je jednou z najdôležitejších častí celého procesu testovania použiteľnosti, pretože bez prípravy môže dôjsť k úplne zbytočným výsledkom. Účastníci testovania musia vykonávať reálne úlohy. Musia to byť tie úlohy, ktoré by skutoční užívatelia s produktom naozaj robili.



Obr. 2.1: Příklad riešenia úlohy hry Robomise

2.2.1 Testovací scénár

Pre používateľské testovanie bol vytvorený jeden scenár so 6 úlohami. Tie boli zamerané na hraciu časť aplikácie. Teda či je hráč schopný sa zorientovať v hre, manipulovať s príkazovými blokmi, prejsť úlohou alebo si v zozname úloh vybrať inú úlohu, ako mu ponúka systém.

Testovací scénár sa skladá z týchto častí:

- Prvou časťou scenára bolo predstavenie osoby, ktorú testovací účastník bude reprezentovať. Nakoniec nebol potrebný, pretože testovanie prebiehalo s potenciálnymi používateľmi, ktorí sa nepotrebovali vcítiť do cudzej role.
- Druhou časťou boli otázky týkajúce sa zázemia účastníkov, aby sme výsledky testovania mohli posúdiť na základe pozície a predošlých skúseností používateľa.
- Treťou časťou bolo samotné testovanie.

- Poslednou úlohou testovaného účastníka bolo ohodnotiť hracie prostredie a prípadne navrhnúť zmeny, pri ktorých by sa mu hrou prechádzalo lepšie.

Testovacie úlohy boli pre potreby testovania napísané v slovenčine a ich presné znenie je uvedené v prílohe.

2.2.2 Testovací účastníci

Pri testovaní bol vždy prítomný iba jeden účastník a moderátor, aby bolo dosiahnuté tých najlepších výsledkov. Viacerí účastníci naraz totiž podliehajú dynamike skupiny a niektorí účastníci tak nemusia vysloviť svoj názor alebo postupne konvergujú k spoločnému názoru, čo v konečnom dôsledku vedie k nepresným dátam [7]. Testovanie len s jedným účastníkom v tom istom čase umožní účastníkovi viac sa uvoľniť, prezentovať svoje skutočné názory a moderátorovi umožní určiť, či používateľ dokáže so systémom pracovať samostatne.

Užívateľského testovania webovej hry Robomise sa zúčastnili piati ľudia. Pri výbere účastníkov som sa zamerala na jednu skupinu potenciálnych používateľov, a tou sú programátori. Táto skupina už základné príkazy, ktoré sú používané v aplikácii, pozná. Tým pádom sa účastníci viac sústredia na hráčské prostredie, manipuláciu s príkazmi a ďalšie rušivé elementy pri hre. Je nepravdepodobné, že by nedokončili úlohu z dôvodu, že nevedia správne poskladať príkazy alebo použiť cyklus. Programátorov som vybrala z troch rôznych firiem: SledovániTV, SychrovNET a ModernTV.

2.3 Priebeh testovania

Po príprave testovania sa plynule prešlo na samotné vykonávanie testovania. To prebiehalo v účastníkovom prirodzenom prostredí, na jeho pracovisku. Každý účastník bol vopred oboznámený s nahrávaním testovania. Testovanie prebehlo bez závažnejších problémov. Po skončení každého testovania bol spracovaný záznam do textovej podoby, ktorá bola podkladom pre vyhodnotenie testovania.

2.4 Vyhodnotenie testovania

Po testovaní sa prešlo do etapy spracovania dát. Dáta boli zaznamenávané hlavne počas testovania, po testovaní sa doplnilo iba zopár drobností z nahrávok. Tieto výstupy sú dostupné v textovej prílohe tejto práce.

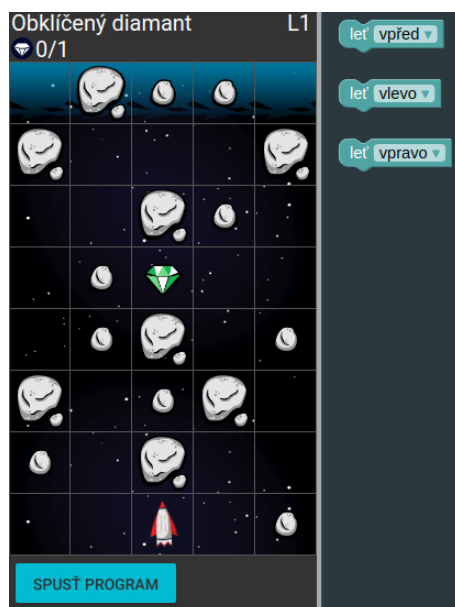
2.4.1 Vyhodnotenie dát

Zo spracovaných dát vyplynulo, že účastníci pochádzajú z rôznych pracovných pozícií. Najväčšie zastúpenie mali frontend programátori, ďalej sa testovania zúčastnil backend developer a UX designer. Ani jeden z nich sa predtým nestretol s podobným typom hry, kde by mohli pomocou blokov reprezentujúcich príkazy ovládať panáčka. Vo väčšine prípadov boli milo prekvapení z tohto grafického programovacieho jazyka.

Tým, že sa pred testovaním účastníci nestretli so žiadnou podobnou hrou na rozvíjanie algoritmického myslenia, mali odlišné očakávania:

- Prvý účastník očakával tlačidlá áno a nie, na základe ktorých bude riešiť algoritmické problémy.
- Druhý tester predpokladal, že uvidí veľa obrázkov a grafov, pri ktorých bude musieť logicky myslieť.
- Tretí účastník očakával veľký problém, v ktorom bude musieť nájsť menšie podproblémy a tie ďalej vyriešiť.
- Štvrtý čakal pythonovskú korytnačku, ktorá bude podľa príkazov kresliť obrázky.
- Poslednému testerovi sa podarilo odhadnúť hru spomedzi účastníkov najviac. Ten si hru predstavoval ako kocky, ktoré sa budú do seba spájať a na základe ich správneho spojenia bude v hre postupovať do ďalších levelov.

Nasledujúca časť sa zaoberá detailným rozborom najväčších problémov, ktoré boli na základe pozorovania odhalené. Každému účastníkovi sa podarilo prejsť všetkých šesť úloh. Počas testovania nenarazili



Obr. 2.2: Úloha Obklíčený diamant

na nič, čo by im bránilo v dokončení úlohy. Nemali problém nájsť plný zoznam úloh, problém im nerobilo ani nájsť začiatok programu, do ktorého mali myšou pretahovať bloky. Niektoré úlohy boli pre testerov náročnejšie a zabralo im viac času ich vyriešiť, ale vo výsledku ich vyriešili bez toho, aby som musela zasiahnuť do testovania.

Na druhej strane som u účastníkov vypožadovala problém s pochopením príkazov: vľavo, vpravo. Napríklad pri úlohe s názvom Obklíčený diamant (viď Obr. 2.2) každý testovací účastník začal skladať príkazy v poradí: štart, vpravo, vpred, vpravo, vpred. Pri poslednom príkaze sa zastavil a oznámil mi, že sa úloha nedá vyriešiť. Potom stlačil tlačidlo Spustiť program, aby zistil, ako sa bude vesmírna loď správať. Vďaka tomu, že hra zvyrazňuje príkaz, kvôli ktorému sa raketka pohla, účastníci pochopili, že príkazom vľavo sa vesmírna loď pohla o jedno pole vyššie a o jedno pole doľava. Potom dokázali správne navoliť príkazy tak, aby nenarazili do asteroidu a získali diamant.

Druhým častým problémom, s ktorým sa stretli 4 účastníci testovania, bolo určenie pozície, v ktorej má raketka zastaviť. Tester

predpokladali, že vesmírna loď má preletieť celým hracím poľom. Takže pri už spomínanej úlohe s názvom Obklúčený diamant zadali 9 príkazov vrátane bloku Štart, pričom príkazov je potreba práve 8. Teda o jeden menej, aby raketka ostala na cieľovej čiare, ktorú reprezentuje modrá farba.

Ďalším problémom, kvôli ktorému nevedelo ľahko vyriešiť úlohu niekoľko testerov, bolo neporozumenie končiacim podmienkam. Vo vyšších leveloch môže raketka zmeniť smer na základe farby, ktorá je pod ňou. Takže napríklad raketka lieta v cykle stále rovno, pokým pod ňou nie je modrá farba. K tomuto problému dochádzalo, pretože som testovacích účastníkov prostredníctvom testovacieho scenára prinútila otvárať úlohy z vyšších levelov, kvôli čomu preskočili vysvetlivky, ktoré túto funkcionality popisujú.

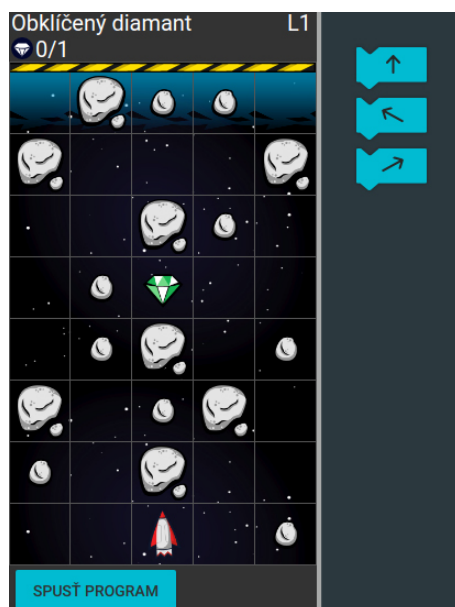
Tretiemu účastníkovi neprišlo intuitívne rozbaľovanie úloh levelu. Takže pri piatej úlohe testovacieho scenára najskôr nevedel, ako spustiť úlohu, ktorá je v cyklickej výzve, po bližšom preskúmaní komponenty na to ale prišiel. Druhý účastník si vybral úlohu, ktorá mala obmedzený počet príkazov, čo je znázornené nad hracou mriežkou ikonou 2.1, a nerozumel, prečo mu bloky stmavli a nemôže ich ďalej používať.

V poslednej fáze testovania účastníci hodnotili aplikáciu na škále od 1 (najhoršie hodnotenie) do 7 (najlepšie hodnotenie). Priemerom hodnotenia piatich testerov je 5, čo ukazuje, že používatelia boli s rozhraním viac spokojní ako nespokojní. Pri otázke, či by si vedeli túto hru predstaviť používať, dvaja odpovedali kladne, jednému vadilo detské prostredie, jeden sa na to pozeral ako na nástroj, ktorým je možné pri pohovoroch vyfiltrovať ľudí bez analytického myslenia. Poslednému sa nepáčilo užívateľské prostredie.

2.4.2 Návrhy riešení

Po uvedení a rozobraní problémov nasleduje časť návrhov možných riešení, ktorými by bolo možné tieto nedostatky eliminovať.

1. Najväčším problémom pre testovacích účastníkov boli nepochopené príkazové bloky: Vpravo, Vľavo. Účastníci navrhli zmeniť názov na Diagonálne vpravo a Diagonálne vľavo. Tomuto názvu ľahko porozumejú programátori, ale obávam sa, že 12-ročné deti by s takýmto názvom mali rovnaký problém ako pred-



Obr. 2.3: Vylepšená úloha Obklíčený diamant

tým testovací účastníci. Preto navrhujem použiť namiesto slov značky: ↖ a ↗, ako je znázornené na obr. 2.3.

2. Problém určenia, kedy má raketka zastaviť, môžeme vyriešiť pridaním zástavky, alebo žltó-čiernej výstražnej pásky na konci hracieho poľa, ako je taktiež zobrazené na obr. 2.3, aby používateľ vedel, kde hra končí a kde už nemôže ísť ďalej.
3. Rozbaľovanie úloh jednotlivých levelov sa dá jednoducho nastaviť, aby bolo defaultne otvorené. Treba však otestovať, či to vyhovuje aj ostatným používateľom.
4. Ďalším návrhom na vylepšenie aplikácie bolo umiestniť legendu príkazov niekde na okraj hry.
5. Programátori sú zvyknutí písať kód v angličtine, preto navrhli pridať možnosť zmeny jazyka aplikácie. V aktuálnej verzii, pokiaľ sú pod českou doménou, musia skladať bloky cyklov: dokud, pokud. Ak by si používateľ mohol zmeniť jazyk na an-

glický, presúval by bloky: while, do-while, ktorým programátor lepšie rozumie.

6. Posledný zaujímavý návrh spočíval v pridaní úvodného videa na domovskú stránku, v ktorom by autori vysvetlili budúcemu hráčovi princíp a bodovanie hry.

Z užívateľského testovania vyplynuli dve hlavné požiadavky na novú natívnu aplikáciu. Prvou požiadavkou je zmeniť bloky vpravo, vľavo. Druhou je zvýrazniť koniec hracieho poľa. Navrhla som vzhľad takto upraveného hracieho pola. Názornú ukážku je možné vidieť na obrázku 2.3. Na mobilnej aplikácii je nutné tento návrh otestovať.

3 Mobilná aplikácia

Nasledujúca kapitola v prvej časti popisuje technológiu použitú pri vývoji novej natívnej aplikácie. V ďalších častiach sa venuje implementačným postupom a funkcionalitám.

3.1 React a React Native

React Native je javascriptový framework pre tvorbu multiplatformových mobilných aplikácií. To znamená, že umožňuje vývojárom vytvárať aplikácie pre iOS i Android. Za jeho vznikom a neustálym vývojom stojí Facebook, Inc. [8] Je založený na javascriptovej knižnici React, ktorá slúži na tvorbu užívateľských rozhraní webových stránok.

React-Native nevyužíva na vykreslenie komponenty služby HTML tagov ako React. Namiesto nich sa používajú špeciálne komponenty poskytované natívnym prostredím. Napríklad namiesto HTML značky `div` (3.1) sa v Native používa komponenta `View` (3.2).

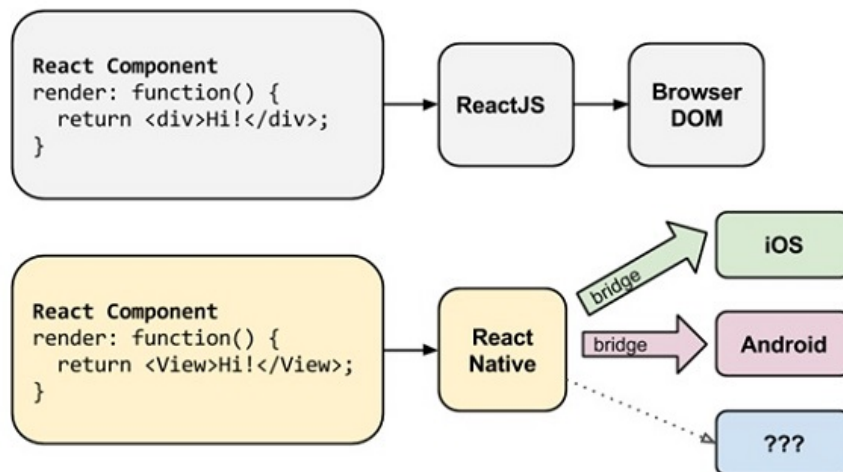
```
1 { const element = (  
2   <div>  
3     <h1>Hello , world!</h1>  
4   </div>  
5 ) ;}
```

Kód 3.1: Základná React komponenta

```
1 { const element = (  
2   <View>  
3     <Text>Hello , world!</Text>  
4   </View>  
5 ) ;}
```

Kód 3.2: Základná React-Native komponenta

V každej aplikácii typicky bežia dva typy vlákien – JavaScript vlákno a natívne vlákna. Natívne vlákna pozostávajú z hlavného vlákna a niekoľkých vlákien bežiacich na pozadí. JavaScript vlákno je špecifické pre React Native. Jeho úlohou je vykonávať javascriptový kód, ktorý sa stará o business logiku aplikácie v samostatnom



Obr. 3.1: Schéma architektúry React Native a porovnanie s ReactJS.
Zdroj:[9]

JavaScript engine¹. Tieto dva typy vlákien spolu nikdy nekomunikujú priamo a ani sa navzájom neblokujú. Je medzi nimi akýsi most (z angl. bridge), ktorý je jadrom React Native.

3.2 Základná kostra projektu

Ako už bolo spomenuté v sekcii 2.1, webová aplikácia je napísaná s použitím Reactu. Mobilná verzia hry Robomise bude naprogramovaná v React-Native. Vďaka tomu, že React aj React-Native sú javascriptové frameworky, môžem znova použiť časť webového kódu. V tejto práci budem písať hlavne o odlišnostiach týchto dvoch implementácií.

Prvý súbor, ktorý sa zavolá, sa volá `Index.js`. V tomto súbore sa nachádza `AppRegistry` trieda, ktorá registruje hlavnú komponentu celej aplikácie.

React-Native aplikácia je zložená z komponent. Komponenta reprezentuje celú obrazovku alebo jej časť. Každá komponenta môže byť zložená z dcérskych komponent. Pre svoje fungovanie potrebuje dáta.

1. JavaScript engine je program alebo interpret, ktorý spúšťa javascriptový kód

Odovzdávanie dát si je možné predstaviť dvoma spôsobmi, pomocou props a state.

Props si je možné predstaviť ako vstup funkcie. Komponenta prijme vstup (props), spracuje ho a zobrazí výstup, nesmie však meniť svoje props [9].

Na dáta, ktoré sú premenlivé, sa používa state. Ten predstavuje stav, v ktorom sa objekt aktuálne nachádza. Komponenta môže svoj stav inicializovať a následne ho aktualizovať kedykoľvek potrebuje.

Napríklad chceme v aplikácii nemenný blikajúci text. V tomto prípade je text props, inicializuje sa iba raz a ostáva nemenný. To, či je text viditeľný alebo nie, sa mení časom. Preto je hodnota vypnutý, zapnutý uložená v state.

Podľa toho, či komponenta pracuje iba s props alebo aj so state, rozdeľujeme komponenty na prezentačné (stateless) a kontajnery (stateful).

Prezentačné komponenty sa zaoberajú tým, ako veci vyzerajú. Popisujú, ako má vyzeráť UI. Prijímajú props z komponent, ktoré ich používajú a na ich základe vykresľujú obsah. V mobilnom projekte sú tieto komponenty v zložke components.

Hlavnou úlohou kontajnerov je starať sa o logiku zobrazovania. Môžu obsahovať prezentačné komponenty, ale aj ďalšie kontajnery. Slúžia ako zdroje dát, ktoré ďalej predávajú svojim komponentám. Kontajnery sú dostupné v zložke containers.

Okrem vyššie spomínaných zložiek sa v mobilnom projekte nachádza súbor package.json, v ktorom je zoznam použitých knižníc, a zložka pages, ktorá obsahuje kontajnery reprezentujúce celé obrazovky. Napríklad súbor PractisePage.js predstavuje celú hráčku plochu hry Robomise. Obsahuje SpaceGameContainer, v ktorom sa raketa pohybuje, a BlocklyEditorContainer, v ktorom používateľ spája príkazové bloky.

3.3 Navigácia

Mobilná aplikácia nie je zložená iba z jednej obrazovky. Preto je potrebné implementovať logiku, ktorá určí, kedy a za akú je potrebné obrazovku vymeniť. Pre navigáciu medzi obrazovkami (pages) je pou-

3. MOBILNÁ APLIKÁCIA

žitá knižnica `react-router-native` ². Túto knižnicu som vybrala pre jej intuitívnosť, schopnosť držať si históriu prechodov a podobnosť s webovou navigáciou. Alternatívnou voľbou bola rozšírená knižnica `react-navigation`.³

```
1 <NativeRouter>
2   <AppContainer>
3     <Switch>
4       <Route exact path="/" component={TasksTableContainer}/>
5       <Route exact path="/tasks" component={TasksTableContainer}/>
6       <Route path="/task/:taskId" component={PracticePage}/>
7     </Switch>
8   </AppContainer>
9 </NativeRouter>
```

Kód 3.3: Router v `index.js`

Použitie routeru v mobilnej aplikácii môžete vidieť v kóde 3.3. Route je najdôležitejšia komponenta knižnice. Jej zodpovednosťou je vykresliť komponentu podľa aktuálnej `path` ⁴. Ak sa v kóde 3.3 `path` rovná `/tasks`, router vykreslí komponentu `TasksTableContainer`. Ak sa cesta zmení na `/task/:taskId`, router vykreslí komponentu `PracticePage`.

`Switch` slúži na vykreslenie prvého dcérskeho `Route`, ktorého `path` je rovnaká ako aktuálna `path` ⁵.

Link poskytuje navigáciu hlbšie v aplikácii. Pomocou parametra `to:String` mení aktuálnu `path`.

3.4 Správa stavov

Užívateľ môže svojím vstupom ovplyvňovať časti aplikácie. Zachovávať konzistenciu a reagovať na tieto zmeny môže byť náchylné k chybám. Tento problém sa snažia vyriešiť state management knižnice, ktoré využívajú jednosmerný tok dát.

2. Dostupná z <https://www.npmjs.com/package/react-router-native>

3. <https://facebook.github.io/react-native/docs/navigation>

4. <https://reacttraining.com/react-router/native/api/Route>

5. <https://reacttraining.com/react-router/native/api/Switch>

Pre správu stavov mobilnej aplikácie bola zvolená knižnica `react-redux`⁶, a to hlavne pre jej veľkú podporu `react` vývojárov. Táto knižnica funguje na princípe globálneho úložiska dát. Toto úložisko je pre aplikáciu jediným zdrojom dát. Tie predáva komponentám. Tieto dáta sú nemenné a pokiaľ ich chceme zmeniť, musíme ich prepísať inou hodnotou.

Komponenty posielajú akcie do `redux`ových `reducerov`, tie na základe informácií z danej akcie zmenia `reduxové` úložisko. Akonáhle sa zmenia dáta v `reduxovom` úložisku, komponenta, ktorej `redux` tieto dáta predáva, na tieto zmeny reaguje.

3.5 Api

Mobilná aplikácia nemá dáta, ako napríklad zoznam úloh, zabudované priamo v sebe. Pre prístup k zdrojom používa `REST API`⁷, ktoré je dostupné na adrese: `https://robomise.cz/learn/api/`. Funkcie, ktoré pracujú s `API`, sa nachádzajú v zložke `api`.

Na `HTTP` požiadavky⁸ je použitá knižnica `axios`⁹. Jej hlavnou výhodou je vykonávanie automatického transformovania `JSON`¹⁰ dát[10]. V aplikácii sú použité dve asynchrónne metódy `axios.get()`, ukážka v kóde 3.4, a `axios.post()`, ukážka v kóde 3.5.

```
1  async get(urlPath) {  
2      let options = await this.getRequestOptions()  
3      let ret = await this.axios.get(urlPath, options)  
4      return ret  
5  }
```

Kód 3.4: Metóda `get` v `api.js`

6. <https://react-redux.js.org/>

7. Representational State Transfer Application Programming Interface

8. HTTP requests

9. <https://www.npmjs.com/package/axios>

10. JavaScript Object Notation

3. MOBILNÁ APLIKÁCIA

```
1  async post(urlPath, data){
2    let options = await this.getRequestOptions()
3    let ret = await this.axios.post(urlPath, data, options)
4    return ret
5  }
```

Kód 3.5: Metóda post v api.js

Vyššie uvedené metódy sa potom používajú vo funkciách v /api/index.js. Napríklad v ukážke 3.6 sa používa vyššie spomenutá metóda get, ktorou v tomto prípade získame informácie o hracích plochách hry.

```
1  export function fetchWorld(url) {
2    let api = Api.getInstance()
3    return api.get(url).then(response => response.data);
4  }
```

Kód 3.6: Funkcia fetchWorld

Problémom bolo, že, aj napriek korektne implementovaným HTTP požiadavkam, mi pri post metóde server vracal: chybový kód 403¹¹.

Problémom bol chýbajúci CSRF¹² token, ktorým sa zaistuje bezpečnosť severu, aby posielali post požiadavky len autorizovaní klienti.

Najskôr som pomocou webových stránok¹³ zistila, ako zahrnúť CSRF token do všetkých HTTP požiadaviek. Ukážka priradenia v kóde 3.7.

```
1  Axios.defaults.headers.common['X-CSRF-TOKEN'] = token;
```

Kód 3.7: Priradenie tokenu do axios

Potom som chcela využiť elegantné riešenie¹⁴, ktoré bolo založené na použití knižníc axios-cookiejar-support a tough-cookie. Bohu-

11. Forbidden error - Nepovolený prístup

12. Cross-site Request Forgery

13. <https://github.com/axios/axios/issues/2024>

14. Dostupné na <https://github.com/axios/axios/issues/943#issuecomment-378219600>

žial som stále naspäť dostávala chybový kód 403 . Pri použití výpisu do konzoly som zistila, že token je naďalej prázdny.

Všimla som si, že v response¹⁵ požiadavky get sú headers, ktoré obsahujú hodnotu žiadaného CSRF tokenu. Ten som pomocou regexu¹⁶ získala a priradila do `Axious.defaults`. Kód 3.8.

```
1 let x = await this.axios.get("/", this.requestOptions);
2 let regex = /(?!<=csrftoken=)\w+(?!=;)/;
3 this.token = regex.exec(x.headers["set-cookie"])[0][0]
```

Kód 3.8: Získanie CSRF tokenu

Pre overenie som znova vypísala token do konzoly. Tentokrát bol už správne nastavený, ale server mi stále vracal kód 403. Potom som prišla na to, že je potreba nastaviť ešte `Referer`¹⁷. Po jeho nastavení na adresu `https://robomise.cz` post požiadavka prebehla úspešne.

3.6 Štýly

Na úpravu vzhľadu a rozloženia komponent sú určené štýly. V React-Native sa používajú `Stylesheets` jazyka JavaScript. Tieto `Stylesheets` sa podobajú kaskádovým štýlom používaným na webových stránkach¹⁸.

Vo webovej aplikácii Robomise mohli vývojári používať štýlové funkcie: `display: block`, `display: table`, `backgroundSize: '100% auto'`, ukážka kódu 3.9, ale v React-Native tieto funkcie chýbajú alebo sú pozmenené. Preto ich bolo nutné v celej aplikácii prepísať. Príklad prepísaného štýlu komponenty `SpaceBackgroundGrid` je v kóde 3.10.

```
1 const backgroundGridStyle = {
2   display: 'table',
3   borderCollapse: 'collapse',
4   backgroundImage: 'url(${spaceBackgroundPath})',
5   backgroundSize: '100% auto',
```

15. Odpoveď

16. Regulárneho výrazu

17. URI, z ktorej bola webová stránka navštívená

18. CSS - Cascading Style Sheets

```
6      backgroundColor: '#112',});
```

Kód 3.9: Štýl webovej komponenty SpacebackgroundGrid

```
1  const styles = StyleSheet.create({
2    fieldStyle : {
3      flex: 1,
4      borderColor: '#555',
5      backgroundColor: 'transparent',
6      borderWidth: 1,
7    },
8    backgroundGridStyle : {
9      flex: 1,
10     backgroundColor: 'transparent',
11   },
12  });
```

Kód 3.10: Štýl natívnej komponenty SpacebackgroundGrid

Aby sa komponenta dynamicky zmenšovala a rozbaľovala na základe toho, koľko má voľného miesta, sa používa `flex`¹⁹. Najčastejšie sa používa `flex: 1`. Vďaka tejto štýlovej funkcii sa komponenta rozťahne do celého dostupného miesta. Ak má vyššie popísaná komponenta dve dcérske komponenty a obidve majú nastavené `flex` na jedna, každá dcérska komponenta bude zaberáť jednu polovicu obrazovky. Dokopy teda zaberajú celé miesto rodičovskej komponenty.

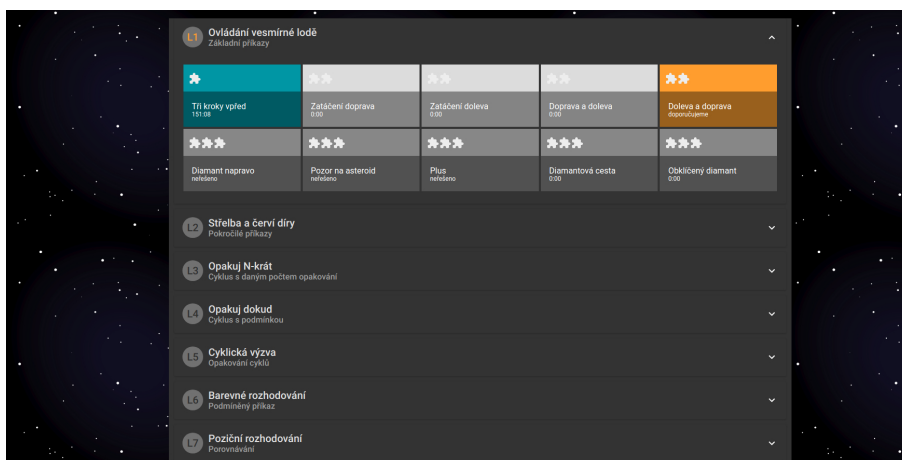
To, či si komponenty rozdelia miesto horizontálne alebo vertikálne, určuje parameter `flexDirection`. Východiskovým nastavením je vertikálne rozdelenie. Ďalšie nastavenia obrazovky sú dostupné na adrese: <https://facebook.github.io/react-native/docs/layout-props>.

Ďalšou odlišnosťou v štylování sú obrázky. Vo webovom rozhraní je možné používať obrázky v rámci štýlov, ukážka 3.9. V React-Native sú obrázky samostatné komponenty, viďte 3.11.

```
1 <Image source={backgroundImagePath} style={styles.fieldStyle}/>
```

Kód 3.11: Obrázková komponenta vo FieldBackground

19. <https://facebook.github.io/react-native/docs/height-and-width>



Obr. 3.2: Webová komponenta TaskTable

3.7 Komponenty hry Robomise

Táto sekcia sa venuje trom základným komponentám hry Robomise: TaskTable, BlocklyEditor, SpaceGame.

3.7.1 TaskTable

Tasktable je prvá komponenta, s ktorou sa používateľ v aplikácii stretne. Jej úlohou je ukázať používateľovi zoznam úloh, práve odporúčanú úlohu, vyriešené úlohy, ešte nevyriešené úlohy a čas riešenia úloh. Ukážku webovej verzie komponenty nájdete na obrázku 3.2.

V ideálnom prípade by používateľ počas hry nemal vstupovať do tohto zoznamu. Dôvodom je nástroj Odporúčanie, ktorý pre hráča vždy vyberie tú najvhodnejšiu úlohu, a to na základe jeho prechodov dokončených úloh²⁰.

Používateľ sa do tejto komponenty dostane pri štarte aplikácie alebo pri stlačení tlačidla ALL TASKS zo SpaceGame²¹ komponenty.

Pri implementácii sa pamätalo na používateľský návrh číslo 3 z kapitoly 2.4.2 a úlohy sa nechali rozbalené. Pretože sa testovací účastníci nesťažovali na mriežku úloh, toto rozloženie bolo ponechané aj v novej

20. Implementačné podrobnosti algoritmu si môžete prečítať v diplomovej práci[6]

21. Prípadne stlačením tlačidla z CompleteTaskDialog

3. MOBILNÁ APLIKÁCIA

Spaceship Control		
Basic Commands		
Thrust Back Forward 1/1/55	Turning Right 1/5/18	Turning Left not tackled
Turning Right And Left 5/4/5	Turning Left And Right 5/5/6	Diamond On Right 0/0/0
Reverse Of Forward 1/5/7	Plus recommended	Diamond Path 1/5/5
Surrounded Diamond 0/0/0		
Shooting and Wormholes		
Advanced Commands		
Shot not tackled	Shooting not tackled	Wormhole Dams not tackled
Tunnel not tackled	Don't Forget Shot not tackled	2diamonds 2wormholes not tackled
Left Shot not tackled		
Repeat N-times		
Keep with Number of Transitions		
Ladder 0/0/0	Diamonds in Messured Cloud 0/0/0	Big Left Turn not tackled

Obr. 3.3: Mobilná komponenta TaskTable

aplikácii. Z toho istého dôvodu bolo ponechané farebné označenie úloh. Oranžová na odporúčanú úlohu, modrá na vyriešenú úlohu a sivá na nevyriešenú úlohu.

Vstupom TaskTable komponenty je dvojrozmerné pole, ktoré reprezentuje levely a ich úlohy. Pre prehľadnosť je každý level v jednom Card. To je komponenta z knižnice `react-native-material-cards`. V jej dcérskej komponente `CardTitle` sa nastavuje názov levelu, podtitulku a farba pozadia.

Ďalšou dcérskou komponentou je `CardContent`. Tá v sebe drží komponentu `GridList` z knižnice `react-native-grid-list`, v ktorej sú uložené `View` s jednotlivými úlohami levelu. Aby sa správali ako tlačidlo, sú ešte obalené komponentou `TouchableOpacity`. Farba sa nastavuje vo `View` vrstve. Ukážku výslednej komponenty je možné vidieť na obrázku 3.3.

3.7.2 SpaceGame

`SpaceGame` je najhlavnejšou komponentou celej hry. Vďaka nej sa používateľ rozhoduje, ako za sebou naskladá riadiace bloky. Nachádza sa tu tlačidlo `Run`, ktorým hráč spúšťa vykonávanie poskladaných

blokov v komponente `BlocklyEditor`. Ďalej komponenta `SpaceGame` obsahuje hracie pole, na ktorom sú rozložené objekty ako raketka, asteroid, meteroid, červie diery, diamanty. Treťou úlohou komponenty je poskytnutie dôležitých informácií, ako:

1. level, v ktorom sa hráč nachádza,
2. maximálny počet výstrelů,
3. počet diamantů, ktoré musí pozbierať,
4. maximálny počet blokov, ktoré môže použiť.

`SpaceGame` reprezentuje konkrétnu úlohu, ktorú si hráč vyberie stlačením jej plochy v `TaskTable`²².

Z používateľských návrhov vyplynulo, že túto komponentu chcú upraviť tak, aby bolo na prvý pohľad jasné, že s raketou nemôžu preletieť posledný riadok hracieho pola, konkrétne ide o návrh číslo 2. To sa pri implementácii aplikovalo.

`SpaceGame` je zložená z troch dcérskych komponent: `GameStatus`, `SpaceWorld` a `GameControls`.

`GameStatus` ukazuje základné informácie o hre. Používateľ na prvý pohľad vie, koľko diamantů musí pozbierať, v ktorom leveli sa nachádza alebo ako sa volá úloha, ktorú práve rieši. Ďalšie údaje sú popísané vyššie pri tretej úlohe komponenty.

Každá informácia, ktorá súvisí s obmedzeniami hry, sa skladá z ikony a textu. Tie sú obalené `View`, ako je možné vidieť v kóde 3.12.

```

1  <View>
2    <Icon name="energy" style={iconStyle}/>
3    <Text style={{textAlign: "center", marginTop: 8}}>
4      {energy.current}/{energy.full}
5    </Text>
6  </View>

```

Kód 3.12: Informácia o možných výstreloch

`SpaceWorld` tvorí mriežka, v ktorej sa raketka pohybuje. Vstupom tejto komponenty je dvojrozmerné pole. Z neho sa zistí, koľko riadkov

22. Prípadne stlačením `NEXT` v `CompleteTaskModal`

a stĺpcov bude mať mriežka `SpaceWorld`. Ďalej sa z tohto poľa vypreparujú objekty a farby jednotlivých políčok. Pod objektami sú myslené: meteroid, asteroid, diamant, červia diera.

Na vytvorenie mriežky sa používa knižnica `react-native-easy-grid`. Tá obsahuje komponenty: `Grid`, `Row`, `Col`. Pomocou komponenty `FieldBackground` sa vyplní jedno políčko mriežky farbou, ktorá mu patrí. Ukážka kódu vytvorenia mriežky nájdete v 3.13.

```
1 <Grid style={styles.backgroundGridStyle}>
2   {backgroundColors.map((backgroundsRow, index) =>
3     <Row style={{height: fieldSize}} key={index}>
4       {backgroundsRow.map((background, bgIndex) =>
5         <Col style={{width: fieldSize}} key={bgIndex}>
6           <FieldBackground key={bgIndex} color={background}
7             size={fieldSize} />
8         </Col>
9       )}
10    </Row>
11  )}
12 </Grid>
```

Kód 3.13: Vytvorenie mriežky `SpaceWorld`

`GameControls` sú tlačidlá, ktorých hlavnou úlohou je spustenie alebo resetovanie hry. Vstupom tejto komponenty je jednorozmerné pole `controls` a funkcia `onClick`. V jednorozmernom poli sú informácie o tom, ktoré tlačidlá sa majú zobrazíť. Tie sa potom vykreslia pomocou komponenty `RaisedTextButton` z knižnice `react-native-material-buttons`. Pri stlačení tlačidla sa vykoná funkcia `onClick` zo vstupu komponenty.

3.7.3 BlocklyEditor

`BlocklyEditor` bol najnáročnejšou komponentou, ale bolo nutné ju naprogramovať, pretože bez nej by nebolo možné hru hrať. Hlavnou úlohou tejto komponenty je sprostredkovať používateľovi bloky, ktoré potom ukladá do správneho poradia. Pri stlačení tlačidla `Run` z komponenty `SpaceGame` sa raketka na ich základe pohybuje.

Blockly je grafický programovací editor založený na systéme „drag & drop“²³. Dovoľuje používateľovi skladať príkazy do programov s tým, že sa nemusí trápiť nad syntaxou alebo pozerať na blikajúci kurzor v príkazovom riadku [11].

Vo webovom projekte sa používa na vytvorenie komponenty knižnica `react-blockly-component`. Tá nemá náhradu v natívnej verzii, pretože samotné Blockly je závislé na SVG²⁴ a DOM²⁵. To znamená, že Blockly môže fungovať iba v prehliadači [12]. Preto musela byť v natívnej aplikácii použitá komponenta `WebView`.

Veľmi nápomocná bola mobilná aplikácia `app-sense`²⁶, ktorá je dostupná na Githube. V tejto mobilnej aplikácii používajú Blockly, takže som sa inšpirovala ich implementačnou logikou.

`WebView` je natívna komponenta, ktorá zobrazuje webový kontent v natívnom prostredí. Jej základnou metódou (`props`) je `source`. Táto metóda načítava statické HTML²⁷ alebo URI²⁸ do `WebView`. V mobilnej `RoboMise` sa do `source` predáva základná kostra HTML stránky, ktorá je v projekte dostupná pod názvom `index.html`. Tá obsahuje skripty pre základné fungovanie Blockly, ktoré boli prevzaté z projektu `app-sence`. V body značke sú dva tagy. Tag `div` reprezentuje plochu `WebView` a vo vnútri značky `xml` sa skladajú bloky. Ukážka značky `body` v kóde 3.14.

```

1 <body>
2   <div id="blocklyDiv" style="height: 100vh; width: 1000px;"
3     ></div>
4   <xml id="toolbox" style="display: none"></xml>
5 </body>
```

Kód 3.14: `body` v `index.html`

Ak si tento kód skompilujeme, na obrazovke tabletu sa nám zobrazí prázdna stránka, nie sú v nej viditeľné žiadne bloky. V projekte `app-sence` majú uprostred značky `xml` už namodelované bloky, preto

23. uchop a pusti

24. Scalable Vector Graphics

25. Document Object Model

26. <https://github.com/sonnylazuardi/app-sense>

27. Hypertext Markup Language

28. Uniform Resource Identifier

sa v ich prípade na obrazovke objavlia. Problémom je, že v hre Robo-mise sa bloky nezobrazujú staticky, ale dynamicky. Každý level má iný typ a počet blokov. To, aké bloky sa majú vygenerovať, sa zisťuje pomocou metódy `getToolboxForTask`, ktorej atribúty sú stav a úloha. Výstupom je pole, v ktorom sú ID jednotlivých blokov. Z tohto poľa sa vytvorí pole JSON²⁹ objektov pomocou metódy `expandBlocks`. Tá navyše vytvorí z jedného bloku typu `fly` tri bloky: vpravo, vľavo a vpred. Kompletný zoznam blokov je v kóde 3.15.

```
1 { type: 'fly ',  
2   fields: { direction: 'ahead' }, },  
3 { type: 'fly ',  
4   fields: { direction: 'left' }, },  
5 { type: 'fly ',  
6   fields: { direction: 'right' }, },  
7 { type: 'shoot' },  
8 { type: 'repeat' },  
9 { type: 'while' },  
10 { type: 'color' },  
11 { type: 'position' },  
12 { type: 'if' },  
13 { type: 'if-else' },
```

Kód 3.15: JSON objekty všetkých blokov

K xml s ID "toolbox", do ktorého sa vkladajú bloky vo formáte XML, sa dá dostať pomocou metódy `document.getElementById("toolbox")`. V React-Native nie je možné túto metódu použiť, pretože nepodporuje DOM.

WebView má metódu `injectJavaScript`. Tá pri načítaní stránky vkladá do stránky JavaScript³⁰, ktorý dostane ako textový vstup.

V ukážke kódu 3.16 pridáme blok typu `repeat` do toolboxu a inicializujeme `demoWorkspace`³¹, ktorý bude potrebný v neskoršej časti projektu na získanie výstupného xml. Po skompilovaní a spustení kódu je na obrazovke tabletu vidieť jeden blok typu `repeat`.

Vyššie spomínaný textový vstup, ktorý sa vkladá do stránky, je dostupný v súbore `BlocklyScreen`.

29. JavaScript Object Notation

30. <https://facebook.github.io/react-native/docs/webview>

31. <https://developers.google.com/blockly/guides/configure/web/fixed-size>

```

1 var toolbox = document.getElementById("toolbox");
2 toolbox.innerHTML = '<block type= "repeat"></block>';
3 var demoWorkspace = Blockly.inject("blocklyDiv", {
4   media: "./media/",
5   toolbox: document.getElementById("toolbox")
6 });

```

Kód 3.16: Inicializácia demoWorkspace

V ďalšej časti sa do toolboxu priradia reálne dáta z aplikácie. K dispozícii je pole JSON objektov, ale toolbox prijíma XML, ktoré je obalené stringom. Preto som vytvorila metódu `generateToolboxXml`, dostupnú v súbore `blocklyXmlGenerator`. Táto metóda vyparsuje dáta a vytvorí z nich XML. To sa priradí do `toolbox.innerHTML`. Ukážka je k dispozícii v kóde 3.16. Po spustení kódu stále nie je vidno požadované bloky, pretože Blockly má iba zopár preddefinovaných blokov. Ak programátor potrebuje iné bloky, musí si ich sám vytvoriť [13].

```

1 Blockly.Blocks['while'] = {
2   init: function() {
3     this.appendValueInput("test")
4       .setCheck("Boolean");
5     this.appendField("While");
6     this.appendStatementInput("body")
7       .setCheck(null);
8     this.setPreviousStatement(true, null);
9     this.setNextStatement(true, null);
10    this.setColour(230);
11    this.setTooltip("");
12    this.setHelpUrl("");
13  }
14 };

```

Kód 3.17: Definícia bloku While

Na vytvorenie vlastného bloku som použila webový nástroj Block Factory³². Postup vytvorenia jedného bloku v tomto nástroji je nasledujúci:

1. Nastaví sa meno bloku. V príklade 3.17 je to `while` v hranatých zátvorkách.

32. <https://blockly-demo.appspot.com/static/demos/blockfactory/index.html>

2. Určíme, ktorými koncami sa blok bude napájať na ďalšie bloky. V ukážke sú to príkazy `setPreviousStatement` a `setNextStatement`, ktoré vyjadrujú, že blok bude mať napojenie zhora aj zdola.
3. Nastavia sa vstupy bloku. V ukážke je to premenná `test`, ktorá na vstupe prijíma hodnoty typu `Boolean`, a premenná `body`, ktorá prijíma hodnoty ktoréhokoľvek typu. Blok nemusí mať žiadny vstup. Príkladom je blok `fly`, ktorý má iba názov (nastavuje sa pomocou funkcie `appendDummyInput`).
4. Ďalším voliteľným krokom je nastaviť farbu bloku. Tá sa nastavuje pomocou funkcie `setColour`.
5. Na konci stačí iba skopírovať vygenerovaný kód v pravej časti webového nástroja a vložiť do projektu. V prípade mobilnej aplikácie Robomise do `Stringu`, ktorý sa vkladá do metódy `InjectedJavaScript`.

Opakovaním vyššie spomínaného postupu som pridala funkcie všetkých blokov spomínaných v ukážke 3.15. Po skompilovaní sa konečne ukázali všetky bloky, ktoré patrili k vybranej úlohe. Po prepnutí na inú úlohu z odlišného levelu sa však bloky neobnovili na nové. K tomu slúži funkcia `updateToolbox[14]`, ktorá sa zavolá na definovanom `demoWorkspace` z ukážky 3.14.

V ďalšom kroku je potrebné spraviť komunikáciu prostredia s `WebView`. Do `Stringu`, ktorý sa vkladá do `WebView`, sa pridá event listener dokumentu³³. Ukážka kódu 3.18. Na strane prostredia sa vytvorí funkcia s názvom `pushXMLChange`, ktorá na `WebView` zavolá funkciu `postMessage` s novým XML pre `Toolbox`, ukážka 3.19. Táto vytvorená funkcia sa zavolá vždy, keď sa prepne úloha. To sa dosiahne pridaním metódy `onLoad` do `WebView`, ktorá má ako parameter vyššie spomínanú funkciu.

33. <https://stackoverflow.com/questions/41160221/react-native-webview-postmessage-does-not-work>

```

1 document.addEventListener("message", function(data) {
2     demoWorkspace.updateToolbox('<xml>' + data.data + '</
  xml>');
3 });

```

Kód 3.18: Obnovenie Toolbox

```

1 pushXMLChange = () => {
2     this.webview.postMessage(this.props.initialXml)
3 };

```

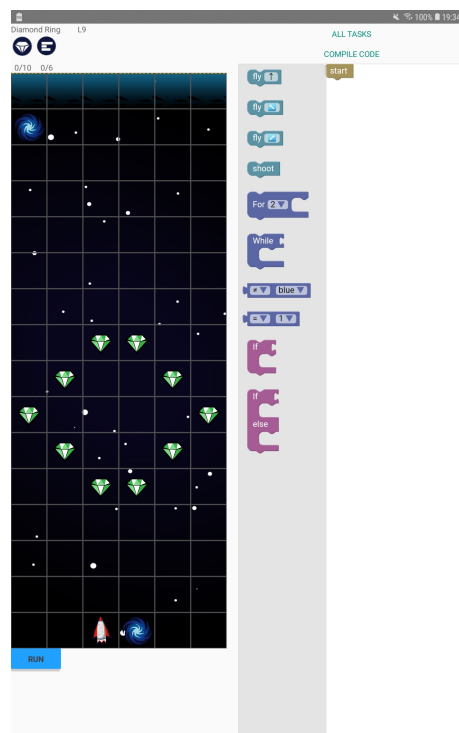
Kód 3.19: Informovanie WebView o zmene XML

Po naimplementovaní vyššie spomínaných funkcií sa pri výbere inej úlohy WebView korektne obnoví a zobrazia sa správne bloky. V tomto štádiu je možné blokmi hýbať, skladať ich za seba, ale pri stlačení tlačidla RUN z komponenty SpaceGame sa raketka nepohne, pretože nevie, ako sú bloky uložené.

V ďalšom kroku je nutné spraviť komunikáciu komponenty WebView s jej okolím. Najskôr som použila `ChangeListener` objektu `demoWorkspace`. V tomto variante komponenta `WebView` poslala XML poskladaných blokov komponente `BlocklyEditor` vždy, keď ich používateľ zmenil. Tento variant som však nemohla použiť, pretože to veľmi spomalilo aplikáciu. Pohyby blokov boli trhané, sekali sa, nemohla som ich napojiť na ďalšie bloky, jednoducho sa hra stala nehrateľnou. Preto som zvolila druhý spôsob.

Nad komponentou `WebView` som vytvorila tlačidlo s názvom "Compile code", čo v slovenčine znamená skompilovať kód, pretože aj v reálnych programoch sa kód najskôr skompiluje a až potom sa vykoná. Takto si hráč aspoň uvedomí, že tam existuje medzikrok. Pri stlačení tlačidla sa pomocou metódy `postMessage` informuje komponenta `WebView`, že má poslať požadované dáta. Komponenta `WebView` dáta spracuje a pomocou metódy `postMessage` ich pošle materskej komponente `BlocklyEditor`. Ukážku získania dát z `demoWorkspace`, spracovania a poslania nájdete v kóde 3.20. Prijemca dát sa nastavuje v `props` `WebView` nazývanej `onMessage`, ktorej parameter je funkcia `receiveAst`. Tá dáta spracuje a pošle komponente `BlocklyEditor`.

3. MOBILNÁ APLIKÁCIA



Obr. 3.4: Výsledné prostredie úlohy mobilnej aplikácie

```
1 var xml = Blockly.Xml.workspaceToDom(demoWorkspace);  
2 const roboAst = blocklyDomToRoboAst(xml);  
3 if(roboAst.body.length) {  
4   window.postMessage(JSON.stringify(roboAst));  
5 }
```

Kód 3.20: Odpoveď WebView na stlačenie tlačidla

Tieto dáta sú vo formáte JSON. Pôvodne sa mali posilať vo formáte XML, ale React-Native nevie s DOM pracovať. Preto ich pomocou metódy `blocklyDomToRoboAst` prekonvertujem ešte vo vnútri komponenty `WebView`. Všetky funkcie na parsovanie XML dát sú dostupné v súbore `BlocklyScreen` v `Stringu`, ktorý sa vkladá do `WebView`. Týmto spôsobom sa bloky nezasekávajú a hra je hrateľná. Pri spustení aplikácie v tomto štádiu môže používateľ skladať bloky za sebou a po skompilovaní a stlačení tlačidla `RUN` sa raketka podľa nich pohybuje.

Takže komponenta `BlocklyEditor` sa skladá z `WebView`, v ktorom hráč skladá bloky, z tlačidla `Compile code`, ktoré uloží obsah blokov, a z tlačidla `All tasks`, ktorým sa používateľ dostane do prehľadu úloh. Ukážku celého prostredia jednej úlohy je možné vidieť na obrázku 3.4. V pravej časti sa nachádza komponenta `BlocklyEditor`.

3.8 Postrehy z vývoja

Mobilnú aplikáciu *Robomise* som začala programovať bez predošlých znalostí `Reactu` alebo `JavaScriptu` všeobecne. Po prečítaní tutoriálu od tvorcov³⁴ som sa naučila základné princípy, určite ho odporúčam každému, kto s týmto jazykom začína. Najväčší problém som mala s pochopením stavov a navigácie.

Vývoj aplikácie veľmi spomaľovali málo hovoriace chybové hlášky. Príklad je na obrázku 3.5. Tie sa zobrazovali priamo na zariadení, na ktorom bežala aplikácia.

Keď sa takáto chybová hláška objavila, veľmi mi pomáhal nástroj `Remote debugger`³⁵ a logy v aplikácii. Tak som sa dostala na miesto, kde sa problém vyskytoval, a odstránila som ho. Často to boli "hlúpe" chyby typu zabudnutej zátvorky alebo zlej verzie knižnice. S tým, že `build` aplikácie prebehol úspešne, pričom v kóde bola zátvorka navyše, som sa stretla prvýkrát.

Naopak, výrazného zrýchlenia vývoja aplikácie som dosiahla vďaka nástroju `Hot Reloading`. Ten umožňuje rýchle načítanie zmien v aplikácii bez nutnosti ďalšieho kompilovania kódu. Jeho základnou myšlienkou je udržať aplikáciu spustenú a predať jej nové verzie súborov, ktoré boli editované počas jej behu, s tým, že sa zachová jej stav.

`React-Native` je stále ešte veľmi mladý jazyk³⁶ a na jeho vývoji sa neustále pracuje. To sa odráža na tom, že potrebné komponenty buď úplne chýbajú, sú implementované iba čiastočne, alebo k nim chýba dokumentácia.

Príkladom je webová knižnica `react-blockly-component`, ktorá v `React-Native` úplne chýba, alebo knižnica `JS-Interpreter`³⁷. Tú

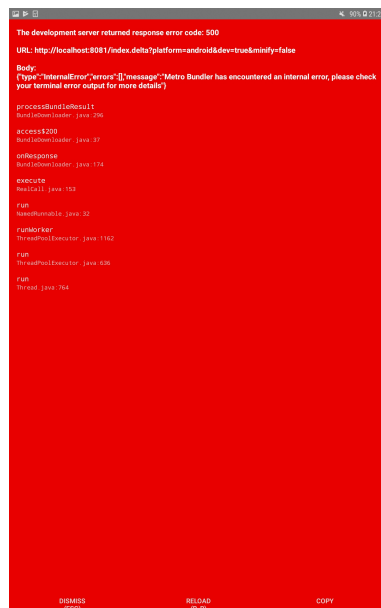
34. <https://facebook.github.io/react-native/docs/getting-started>

35. <https://facebook.github.io/react-native/docs/debugging>

36. Začiatok tohto frameworku sa datuje od januára 2015

37. <https://github.com/NeilFraser/JS-Interpreter>

3. MOBILNÁ APLIKÁCIA



Obr. 3.5: Príklad málo hovoriacej chybovej hlášky v React-Native

som musela forknúť a modifikovať, aby sa mohla používať v React-Native. Táto knižnica sa v projekte používa na prekonvertovanie výstupu `WebView`³⁸ do JavaScriptu. Zmenu v tejto knižnici môžete vidieť v ukážke kódu 3.22.

```
1 this['Interpreter'] = Interpreter
```

Kód 3.21: Riadok v knižnici JS-Interpreter

```
1 module.exports.Interpreter = Interpreter
```

Kód 3.22: Náhrada riadku z ukážky 3.21

38. Výstupom je JSON

4 Používateľské testovanie mobilnej aplikácie

Pre používateľské testovanie mobilnej aplikácie bol použitý rovnaký testovací scenár ako pri webovom používateľskom testovaní. Tento testovací scenár je popísaný v kapitole 2.2.1. Nový testovací scenár sa nevytváral, pretože na pôvodný tester reagovali dobre a zistili sa vďaka nemu nedostatky aplikácie. Ďalším dôvodom bolo zabezpečenie rovnakých podmienok pri testovaní.

Testovania sa zúčastnili piati účastníci. Všetci boli programátori. Dvaja z nich boli účastníci webového testovania. Tí neodpovedali na otázky ohľadom testovacieho zázemia, pretože by ich odpovede boli rovnaké ako pri webovom testovaní.

Pri testovaní bol vždy prítomný iba jeden účastník a moderátor, ako to bolo aj pri webovom testovaní. Samotné vykonávanie testovania prebiehalo v účastníkovom prirodzenom prostredí, na jeho pracovisku. Tentokrát sa nenahrávalo. Testovanie prebehlo bez závažnejších problémov. Po skončení každého testovania bol spracovaný záznam do textovej podoby. Tá je podkladom pre vyhodnotenie testovania.

4.1 Vyhodnotenie dát

Zo spracovaných dát, ktoré sú dostupné v textovej prílohe tejto práce, vyplynulo, že účastníci pochádzajú z rôznych pracovných pozícií. Testovania sa zúčastnili traja frontend programátori a dvaja backend programátori.

Z troch nových účastníkov sa jeden stretol s podobným typom hry. Konkrétne išlo o hru "Robotanik"¹, ktorá slúži na precvičovanie rekurzie. Tento tester očakával podobné hracie prostredie. Ďalší dvaja skôr očakávali vkladanie kúskov kódu do predpripravenej šablóny alebo programovacie hádanky.

Nasledujúca časť sa zaoberá rozborom najväčších problémov, ktoré boli na základe pozorovania odhalené. Účastníci počas testovania nenarazili na nič, čo by im bránilo v dokončení úlohy. Všetky vyriešili bez toho, aby som musela zasiahnuť do testovania.

1. <https://www.umimeprogramovat.cz/robotanik>

Najväčším problémom zo všetkých bolo tlačidlo "Compile code". Väčšina testerov si ho hneď nevšimla. Zaznamenali ho až vtedy, keď stlačili Run a raketka sa nepohla napriek tomu, že mali poskladanú sekvenciu blokov.

Ďalším nepríjemným problémom bolo sekanie komponenty `WebView`. To sa prejavilo pri dlhšom riešení úlohy a niekoľkonásobnom kompilovaní kódu. Ak používateľ niekoľko sekúnd nehýbal s blokmi, sekanie prestalo, ale aj tak to zanechalo negatívny dojem. Celkovo je mobilná aplikácia pomalšia ako jej webová verzia. Dlhšie trvá, kým sa načítajú komponenty alebo kým raketka spraví prvý pohyb.

Traja tester sa sťažovali, že blok je veľmi malý a cez prst nevidia, či sa už napojil na predtým poskladané bloky.

Pozitívnym zistením bolo, že ani jeden z účastníkov nemal problémy s blokmi Vpravo alebo Vľavo. Iba jednému testerovi vadilo, že blok Vpred má inú ikonku ako zvyšné letiace bloky.

Vďaka žltó-čiernej páske všetci pochopili, že nemajú s raketkou prelietavať hracie pole.

Na rozbalené úlohy sa nikto nesťažoval, ale podotkli, že by im nevadilo, ak by boli zbalené ako vo webovej aplikácii.

4.2 Návrhy riešení

Po rozobraní problémov nasledujú možné riešenia, ktorými by bolo možné tieto nedostatky eliminovať:

1. Účastníci navrhli, že tlačidlo `Compile code` stačí viac zviditeľniť. Napríklad presunúť ku tlačidlu Run. Ďalším variantom bolo znemožniť stlačenie tlačidla Run, pokiaľ sa neskompiluje kód. Farebne odlíšiť tlačidlo `Compile code` vždy, keď sa zmení sekvencia blokov vo `WebView`.
2. Zmeniť veľkosť príkazových blokov je náročnejšie, pretože ich veľkosti sú nastavené v scriptoch tretej strany. V projekte sa tieto scripty používajú v súbore `index.html`. Jednou z možností je spraviť si vlastné natívne tlačidlá, ktoré už nebudú potrebovať komponentu `WebView`. Mínusom tejto alternatívy je nutnosť na-programovania celej logiky blocky. Plusom je však vyriešenie sekania príkazových blokov a dlhého načítania komponent.

3. Ďalším návrhom na vylepšenie mobilnej verzie hry Robomise je zlepšenie UI, konkrétne viac zafarbiť prostredie alebo použiť jednotné šípky príkazových blokov.
4. Posledným návrhom bolo pridanie legendy príkazov, napríklad pod komponentu SpaceGame.

Na konci testovania účastníci hodnotili použiteľnosť aplikácie na škále od 1 do 7, kde 7 je najlepšie hodnotenie. Priemerom hodnotenia bola hodnota 5, čo je rovnaké číslo ako pri hodnotení webovej aplikácie.

Dvoch účastníkov, ktorí sa zúčastnili obidvoch testovaní, som sa spýtala na porovnanie mobilnej aplikácie oproti webovej. Mobilná aplikácia je lepšia v pochopení príkazov Vľavo a Vpravo, v ohraničení hracieho poľa, v používaní prstov namiesto myši, ale oproti webovej aplikácii je pomalšia a musí sa používať o jeden krok viac, ktorým je kompilácia kódu.

5 Záver

Práca Robomise v mobilnom prostredí se venuje vývoju aplikácie Robomise pre mobilné prostredie OS Android. Východiskom bola webová aplikácia, ktorá nevyhovavala dotykovému ovládaniu.

Webovú aplikáciu som podrobila užívateľskému testovaniu a poznatky z testovania som zapracovala do svojej mobilnej verzie. Išlo predovšetkým o nepochopené príkazové bloky vpravo a vľavo a o ohraničenie hracieho poľa.

Pre vývoj bol použitý React-Native, pretože webová aplikácia bola napísaná v Reacte a mohla som využiť časť jej kódu. Navyše, vďaka React-Native je možné použiť 90 % kódu na podporu iOS platformy[15]. Vďaka tomu sa ušetrí čas a financie, ktoré by sa vložili do vývoja novej platformy.

Výsledná aplikácia v React-Native bola znovu podrobená užívateľskému testovaniu. Implementované vylepšenia oproti webovej verzii boli kladne prijaté a zlepšil sa prechod aplikáciou.

Z užívateľského testovania mobilnej aplikácie vyplynuli návrhy na vylepšenie. Hlavne v oblasti UX, ako napríklad presunutie tlačidla compile.

Aplikácia Robomise má potenciál byť využívaná ako nástroj na rozvoj algoritmického myslenia pre deti alebo ako filtračný nástroj na pohovoroch programátorov. Odporúčam jej ďalší vývoj práve v jazyku React-Native.

A Príloha

A.1 Testovací scénár

Ste uchádzačom na pozíciu programátora do Vašej vysnívanej IT firmy. Ste vynikajúci programátor a algoritmy Vám nerobia žiadny problém. Na pohovore Vám namiesto bežného programovania na papier predložia hru. Vy sa nenecháte vyhodiť z miery a rozhodli ste sa nazbierať čo najviac bodov.

1. Spustíte 1. úlohu a nájdite začiatok programu.
2. Pridajte a odstráňte príkaz.
3. Prejdite 1. úlohou.
4. Vyberte a vyriešte úlohu s názvom Obklúčený diamant.
5. Chodíte do prehľadu úloh, vyberte si úlohu z cyklickej výzvy a vyriešte ju.
6. Chodíte do prehľadu úloh, vyberte si úlohu z farebného rozhodovania a vyriešte ju.

A.1.1 Zázemie účastníkov

Aká je Vaša pracovná pozícia a zameranie? Pracovali alebo videli ste predtým podobné grafické rozhranie ako RoboMise?

A.1.2 Testovacie zázemie

Ako si predstavujete grafické prostredie na overovanie algoritmického myslenia? Čo by ste chceli v tomto grafickom rozhraní vidieť?

A.1.3 Pred testovaním

Momentálne môžete vidieť obrazovku, ktorá sa zobrazí hneď po načítaní stránky. Ja Vás poprosím, aby ste všetky svoje kroky komentovali. Nebojte sa povedať, ak sa Vám niečo nepozdáva. Môžete sa pustiť do plnenia prvej úlohy.

A.1.4 Uzavretie testovania

Máte pripomienky alebo nápady k rozhraniu, ktoré ste práve videli? Na škále od 1 do 7, kde 7 je najlepšie ohodnotenie, ako užitočné je pre Vás toto rozhranie? Vedeli by ste si predstaviť ho používať?

A.1.5 Ciele testovania

1. Identifikovanie toho, či je používateľ schopný dokončiť pripravené úlohy.
2. Zistenie spokojnosti účastníkov s testovaným rozhraním.
3. Zistenie očakávania účastníkov od testovaného rozhrania.
4. Identifikovanie slabých a silných stránok používateľského rozhrania.

A.1.6 Testovacie otázky

1. Ktoré časti a komponenty rozhrania robili účastníkom najväčší problém?
2. Aká je úspešnosť dokončenia úloh?
3. Aké chyby sa počas testovania našli?

A.2 Záznam webového používateľského testovania

A.2.1 Zázemie účastníkov

Aká je vaša pracovná pozícia a zameranie?

1. backend developer
2. frontend developer
3. UX designer
4. frontend developer
5. frontend developer

Pracovali alebo videli ste predtým podobné grafické rozhranie ako RoboMise?

1. nie
2. nie
3. nie
4. nie
5. nie

A.2.2 Testovacie zázemie

Ako si predstavujete grafické prostredie na overovanie algoritmického myslenia?

1. tlačidlá áno, nie
2. veľa obrázkov a grafov, ktoré nútia logicky myslieť, interaktívne
3. veľký problém, ktorý rozloží človek na menšie podproblémy a vyrieši ich
4. korytnačka, ktorá kreslí obrázky podľa príkazov
5. skladanie kociek ako príkazov a plnenie úloh

Čo by ste chceli v tomto grafickom rozhraní vidieť?

1. otázku a odpovede
2. flow diagram
3. skladanie vecí dohromady
4. kreslenie objektu
5. kocky, ktoré sa spájajú

A.2.3 Uzavretie testovania

Máte pripomienky alebo nápady k rozhraniu, ktoré ste práve videli?

1. pridať k úlohám legendu, názorné video skladania kociek, inak označené bloky vpravo a vľavo, inak zvýrazniť cieľovú rovinku
2. zvýrazniť povolený počet príkazov, označiť bloky vpravo a vľavo šípkami, odstrániť možnosť spraviť z kocky vpravo kocku vľavo
3. skladať kocky zľava doprava a nie zhora nadol, inak pomenovať kocky vpravo a vľavo, rozklikávacie kategórie, zvýrazniť koniec hracieho poľa
4. nepoužívať české príkazy, ale anglické, šípky namiesto blokov vpravo a vľavo, vysvetlivky k príkazom
5. príkaz vľavo má ísť doľava a nie hore a doľava, pridať názornú ukážku príkazov, inak označiť cieľovú rovinku

Na škále od 1 do 7, kde 7 je najlepšie ohodnotenie, ako použiteľné je pre Vás toto rozhranie?

1. 5
2. 3
3. 6
4. 5
5. 6

Vedeli by ste si predstaviť toto prostredie používať?

1. Áno, ani by ma to nenahnevalo.
2. Bol by som zmätený z užívateľského prostredia. Musím si to všetko vyskúšať.
3. Viem si to predstaviť na pohovoroch na filtráciu ľudí.
4. Asi áno, len mi to príde príliš detské.
5. Nemám s prostredím problém.

A.3 Záznam mobilného používateľského testovania

A.3.1 Zázemie účastníkov

Aká je Vaša pracovná pozícia a zameranie?

1. backend developer
2. frontend developer
3. frontend developer
4. frontend developer
5. backend developer

Pracovali alebo videli ste predtým podobné grafické rozhranie ako RoboMise?

1. nie
2. áno
3. nie

A.3.2 Testovacie zázemie

Ako si predstavujete grafické prostredie na overovanie algoritmického myslenia?

1. priradovanie kódu do iného kódu
2. niečo na spôsob Robotikona
3. spájanie objektov do seba

Čo by ste chceli v tomto grafickom rozhraní vidieť?

1. šablónu a bloky príkazov, ktoré sa budú priradovať do tejto šablóny
2. robota, s ktorým budem na základe príkazov hýbať
3. objekty, logické funkcie

A.3.3 Uzavretie testovania

Máte pripomienky alebo nápady k rozhraniu, ktoré ste práve videli?

1. Trošku mi prídu zvláštne rozdielne ikonky na šípkach, Compile tlačidlo dať vedľa Run, zakázať spustenie bez kompilácie.
2. Tlačidlo Run by mohlo upozorniť na to, že treba skompilovať kód.
3. Zväčšiť príkazové bloky, zvýrazniť tlačidlo Compile, zrýchliť aplikáciu.
4. Viac zvýrazniť ikonku maximálnej dĺžky kódu, pridať legendu príkazov, pridať informáciu o tom, že kód treba skompilovať, bloky sú príliš malé.
5. Tlačidlo Compile code je málo viditeľné, pridať legendu príkazov.

Na škále od 1 do 7, kde 7 je najlepšie ohodnotenie, ako použiteľné je pre Vás toto rozhranie?

1. 5
2. 6
3. 4
4. 4
5. 6

Vedeli by ste si predstaviť toto prostredie používať?

1. Neviem si veľmi predstaviť, na čo.
2. Áno, páči sa mi a chcem si hru ešte zahrať.
3. Myslím si, že pre syna by to bola skvelá hra.
4. Ak budú bloky väčšie, tak áno.
5. Áno a páči sa mi, že je obmedzený počet príkazov.

Bibliografia

1. *Forecast and Trends* [online]. Cisco, 2018 [cit. 2018-11-07]. Dostupné z: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>.
2. *Reports* [online]. Statcounter, 2018 [cit. 2018-11-07]. Dostupné z: <https://statcounter.com/demo/reports/>.
3. PEIRCE, N.; CONLAN, O.; WADE, V. Adaptive Educational Games: Providing Non-invasive Personalised Learning Experiences. 2008, s. 28–35. ISBN 978-0-7695-3409-1.
4. VAŠKOVÁ, Eva. *Testování uživatelského prožitku nástroje pro HA Cluster* [online]. 2018 [cit. 2019-05-10]. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedúci práce Marek GRÁC.
5. KRUG, Steve. Nenuťte uživatele přemýšlet!: praktický průvodce testováním a opravou chyb použitelnost. *Computer Press*. 2010. ISBN 978-80-251-2923-4.
6. EFFENBERGER, Tomáš. *Adaptabilní systém pro výuku základů programování* [online]. 2018 [cit. 2019-05-10]. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedúci práce Radek PELÁNEK.
7. LORANGER, Hoa. *Checklist for Planning Usability Studies* [online]. 2016 [cit. 2019-05-07]. Dostupné z: <https://www.nngroup.com/articles/usability-test-checklist/>.
8. *Facebook Open Source*. Facebook, Inc., 2019. Dostupné tiež z: <https://opensource.facebook.com/>.
9. EISENMAN, Bonnie. *Writing Cross-Platform Apps with React Native*. 2016. Dostupné tiež z: <https://www.infoq.com/articles/react-native-introduction>.
10. COPES, Flavio. *HTTP requests using Axios*. 2018. Dostupné tiež z: <https://flaviocopes.com/axios/>.
11. *Introduction to Blockly*. Google Developers, 2018. Dostupné tiež z: <https://developers.google.com/blockly/guides/overview>.

BIBLIOGRAFIA

12. BUDIN, Nat. *Can this be running in react native app?* 2018. Dostupné tiež z: <https://github.com/patientslikeme/react-blockly-component/issues/22>.
13. *Add Custom Blocks*. Google, 2018. Dostupné tiež z: <https://developers.google.com/blockly/guides/configure/web/custom-blocks>.
14. *Changing the Toolbox*. Google, 2018. Dostupné tiež z: https://developers.google.com/blockly/guides/configure/web/toolbox#changing_the_toolbox.
15. SINGH, Rahul. *How good is React Native?* 2018. Dostupné tiež z: <https://www.quora.com/How-good-is-React-Native>.