Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andraž Pencelj

Mobilna aplikacija za reševanje križank

diplomsko delo na univerzitetnem študiju

Mentor: prof. dr. Saša Divjak

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01917/2013
Datum: 03.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ANDRAŽ PENCELJ**

Naslov: **MOBILNA APLIKACIJA ZA REŠEVANJE KRIŽANK**
CROSSWORD SOLVING MOBILE APLICATION

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Razvijte mobilno aplikacijo za reševanje križank z ogrodjem PhoneGap za mobilno platformo Android. Najprej predstavite križanke, njihovo zgodovino ter uporabljene tehnologije. Posebno pozornost posvetite ogrodju PhoneGap, ki omogoča razvoj mobilnih aplikacij za več moblinih platform hkrati. Predstavite HTML, CSS in JavaScript s svojima knjižnicama jQuery in jQuery Mobile. Opišite izdelavo domorodnega dela aplikacije, strukture HTML, izdelave menijev in logike. Posebej se posvetite izdelavi in predstavitvi križank, ki so prilagojene mobilnim napravam. Podajte tudi podatkovno bazo, v katero se shranjujejo križanke in podatki o uporabnikih.

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Andraž Pencelj, z vpisno številko **63060259**, sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za reševanje križank

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Saše Divjaka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 27. septembra 2013 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Saši Divjaku za pomoč in napotke pri izdelavi diplomske naloge. Zahvaljujem se tudi Leonu in Dejanu za podane predloge ter Anžetu za lektoriranje diplomskega dela.

Posebej se zahvaljujem svoji družini, ki mi je vsa leta študija stala ob strani in me spodbujala.

Hvala!

Kazalo

Seznam uporabljenih kratic

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	Hyper Text Markup Language
JSON	JavaScript Object Notation
SDK	Software Development Kit
XML	Extensible Markup Language

Povzetek

V diplomski nalogi je opisan razvoj mobilne aplikacije za reševanje križank z ogrodjem PhoneGap. Poudarek je na razvoju aplikacije za mobilno platformo Android. V nalogi ni predstavljen strežniški del aplikacije.

V prvem delu diplomske naloge so predstavljene križanke in njihova zgodovina ter uporabljene tehnologije. Pri uporabljenih tehnologijah je posvečena večja pozornost ogrodju PhoneGap, ki omogoča razvoj mobilnih aplikacij za več mobilnih platform hkrati, in mobilnim spletnim brskalnikom. Na kratko so predstavljeni še HTML, CSS in JavaScript s svojima knjižnicama jQuery in jQuery Mobile.

V drugem delu diplomske naloge je predstavljen celoten razvoj aplikacije. Opisana je izdelava domorodnega dela aplikacije, strukture HTML, izdelava menijev in logika. Posebej se naloga posveča izdelavi in predstavitvi križank, ki so prilagojene mobilnim napravam. Predstavljena je tudi podatkovna baza, v katero se shranjujejo križanke in podatki o uporabnikih.

V zaključku naloge so opisane večje težave, ki so se pojavile pri razvoju aplikacije, in možne izboljšave.

Ključne besede: Android, JavaScript, križanka, mobilna aplikacija, PhoneGap.

Abstract

The thesis describes development of crossword solving mobile application with PhoneGap framework. The emphasis is on application development for Android platform. Thesis describes only application which runs on mobile devices without server-side part.

First part of thesis describes crosswords and their history. It also describes technologies which were used. Emphasis is on describing PhoneGap framework and mobile web browsers. PhoneGap framework is used to develop native applications for multiple mobile platforms simultaneously. This part also presents web technologies: HTML, CSS and JavaScript with jQuery and jQuery Mobile libraries, used through development.

Second part describes application development. It describes native part of application, HTML structure of application, menus and logic. The main part is dedicated to creating crosswords and functionalities that suit mobile devices. This part also describes database, which is used to store crosswords and user data.

At the end are presented and described major problems which were encountered through application development and possible improvements.

Key words: Android, JavaScript, crossword, mobile application, PhoneGap.

Poglavje 1

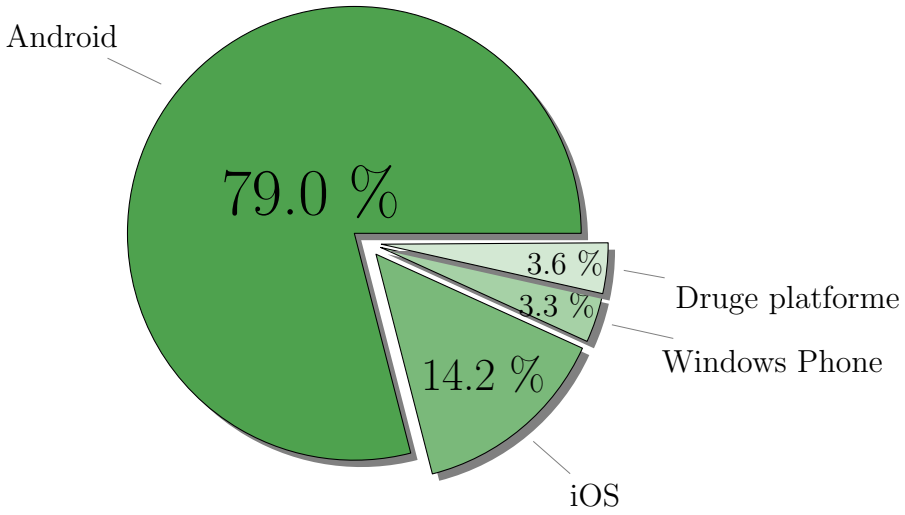
Uvod

Razvijalec mobilnih aplikacij ima na izbiro kar nekaj mobilnih platform (mobilnih operacijskih sistemov), med katerimi mora izbrati tisto, ki bo zaobjela čim večji krog ljudi, ki bodo aplikacijo potrebovali. Na sliki 1.1 je prikazan tržni delež najbolj razširjenih mobilnih platform [2]. V drugem kvartalu leta 2013 so bile najbolj prodajane mobilne naprave s platformo Android in iOS. Njun skupni tržni delež je bil 93,2 %. Smiseln bi bil razvoj za obe platformi, kar pa prinese višje stroške razvoja.

Danes so na trgu prisotna ogrodja (angl. framework), s katerimi je mogoče razviti aplikacijo za različne mobilne platforme hkrati. Aplikacijo razvijemo enkrat in jo distribuiramo na različne platforme. S tem je zajet maksimalno velik krog uporabnikov. Omenjena ogrodja izkoriščajo spletne brskalnike, znotraj katerih se aplikacija izvaja. Aplikacija je razvita s spletnimi tehnologijami HTML, CSS in JavaScript.

V svoji diplomski nalogi sem opisal razvoj mobilne aplikacije z ogrodjem PhoneGap. Aplikacija je sestavljena iz dveh delov. Prvi del je aplikacija, ki se izvaja na mobilni napravi, drugi del pa je strežniški. Slednji v diplomski nalogi ni opisan.

Ne glede na to, da se lahko aplikacija izvaja na več mobilnih platformah, ima vsaka platforma svoje posebnosti, ki so povezane z domorodnim (angl. native) delom aplikacije. Ker sem imel na voljo le mobilne naprave s



Slika 1.1: Tržni delež prodanih mobilnih naprav z različnimi mobilnimi platformami v drugem kvartalu leta 2013.

platformo Android sem v nalogi opisal izdelavo le za to platformo.

Namen aplikacije je prenesti reševanje križank na mobilne naprave. Vključuje dva načina igranja. V enoigralskem načinu igralec rešuje križanke na enak način, kot se to počne s tistimi v tiskani obliki. V večigralskem načinu med seboj tekmujeta dva igralca v reševanju križank tako, da izmenično vpisujeta besede. Tisti, ki vnese več pravilnih besed, je na koncu zmagovalec.

Poglavje 2

Križanke

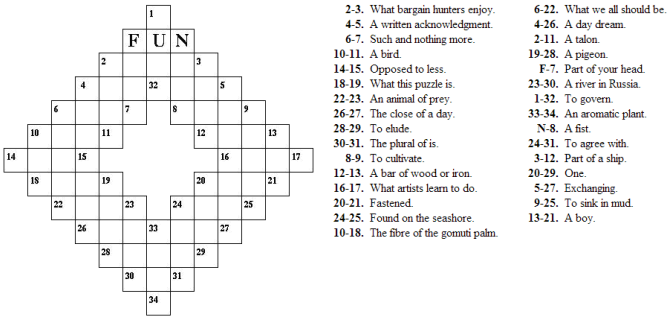
Križanka je besedna uganka, ki je predstavljena z mrežo polnih in praznih kvadratkov (polj). Cilj vsake križanke je zapolniti vse prazne kvadratke s črkami, ki tvorijo smiselne besede. Te besede so vpisi pravilnih odgovorov na vprašanja.

Zgodovina križank sega v leto 1890, ko je bila 14. septembra v reviji Il Secolo Ilustrato della Domenica objavljena prva križanka. Avtor križanke je bil Giuseppe Airoldi. To je bila majhna križanka dimenzij 4x4. Vendar ta križanka ni priznana kot prva in Giuseppe Airoldi ne kot izumitelj križank. Ta čast pripada Arthurju Wyneeju, ki je 21. decembra 1913 v nedeljskem časopisu New York World objavil prvo križanko, podobno današnjim. Križanka je prikazana na sliki 2.1.

V naslednjih letih so se začele križanke uveljavljati kot dodatne tedenske vsebine v mnogih časopisih. Leta 1924 je izšla prva knjiga križank pri založbi Simon and Schuster. Križanke so skozi zgodovino postajale vedno bolj popularne, lahko bi rekli, da so v nekem obdobju zasvojile ljudi.

Križanke se med seboj ločijo po velikosti, obliki, količini črnih (polnih) kvadratkov ter načinu postavitve vprašanj.

Ameriške križanke vsebujejo razmeroma malo črnih kvadratkov. Vprašanja so postavljena ob ali pod križanko. Vsakemu vprašanju pripada številka, ki je vpisana v križanki. Ta označuje, kam mora reševalec vpisati odgovor na



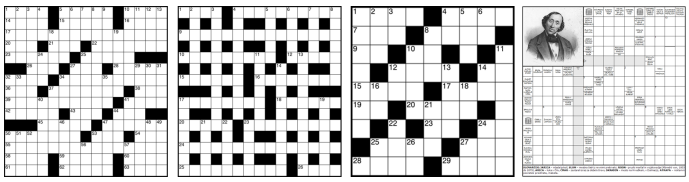
Slika 2.1: Prva sodobna križanka, objavljena v časopisu New York World 21. decembra 1913 [11].

vprašanje. Vprašanja so razdeljena na vodoravna in navpična.

Britanske križanke vsebujejo več črnih kvadratkov kot ameriške. Poleg tega so te križanke simetrične. To pomeni, da je vzorec črnih in vnosnih kvadratkov enak, če križanko zavrtimo za 180 stopinj. Vprašanja so lahko postavljena ob ali pod križanko.

Japonske križanke imajo v kotih vedno vnosne kvadratke. Za črne kvadratke velja pravilo, da se lahko dva črna kvadratka stikata le s kotom in ne s stranico kvadratka. Vprašanja so postavljena ob ali pod križanko.

Švedske križanke so tiste, ki so nam v Sloveniji najbolj poznane. To so križanke, ki pogosto vsebujejo slike in nimajo klasičnih črnih kvadratkov. Namesto črnih kvadratkov imamo v kvadratke napisana vprašanja. S postavitvijo vprašanja je točno določeno, v katere kvadratke je potrebno vpisati rešitev [11]. Slika 2.2 prikazuje opisane vrste križank.



Slika 2.2: Prva slika prikazuje ameriško križanko, druga britansko križanko, tretja japonsko križanko in četrta švedsko križanko [11].

Poglavje 3

Uporabljene tehnologije

Pri izdelavi diplomske naloge sem uporabil razvojno okolje Eclipse z dvema dodatkoma. Prvi dodatek je SDK za platformo Android. Drugi dodatek pa so orodja za delo s spletnimi tehnologijami Eclipse Web Tools. Ogrodje PhoneGap, mobilni spletni brskalniki in spletne tehnologije, ki sem jih uporabil za izdelavo diplomske naloge, so opisani v nadaljevanju.

3.1 PhoneGap

Prostodostopno in odprtokodno ogrodje (angl. framework) PhoneGap omogoča razvoj mobilnih aplikacij s standardnimi spletnimi tehnologijami, kot so HTML, CSS in JavaScript [15]. Glavni namen ogrodja je poenostavitev razvoja aplikacij tako, da ni potreben celoten razvoj za vsako mobilno platformo posebej. Slika 3.1 prikazuje postopek.

Aplikacije, razvite z ogrodjem PhoneGap, sodijo med tako imenovane hibridne aplikacije. To so aplikacije, ki uporabljajo tako domorodni programski jezik platforme kot tudi spletne tehnologije. Uporabniški vmesnik je narejen s HTML-jem in CSS-om, logika in komunikacija s strežnikom pa z JavaScriptom. Tisti deli aplikacije, ki skrbijo za nadzor nad napravo, so narejeni z domorodnim programskim jezikom izbrane platforme.

Ogrodje deluje kot povezava (most) med JavaScriptom in napravo. Omogoča

Slika 3.1: Postopek razvoja aplikacije z ogrodjem PhoneGap. Aplikacijo zapakiramo z ogrodjem PhoneGap in jo distribuirati na različne mobilne platforme [12].

dostop do komponent naprave preko svojih JavaScript API-jev [4]. Trenutno so na voljo API-ji:

- **datoteke:** operacije nad datotekami,
- **dogodki:** dogodki življenjskega cikla aplikacije,
- **globalizacija:** nastavljanje operacij glede na lokacijo naprave,
- **globalno pozicioniranje:** podatki GPS senzorja,
- **kamera:** dostop do kamere,
- **kompas:** smer, v katero je naprava obrnjena,
- **naprava:** podatki o programski in strojni opremi naprave,
- **obvestila:** vizualna in zvočna obvestila,
- **pospeškomer:** podatki o gibanju naprave,
- **povezljivost:** dostop do Wi-Fi in drugih omrežnih povezav,
- **shranjevanje:** shranjevanje podatkov v podatkovno bazo,
- **spletni brskalnik:** dostop do spletnega brskalnika,
- **stiki:** dostop do podatkovne baze stikov,
- **začetni zaslon:** prikazovanje in skrivanje začetnega zaslona,

- **zajem:** zajem videa in zvoka,
- **zvok:** snemanje in predvajanje glasbenih datotek.

Če obstoječi API-ji niso dovolj, se lahko funkcionalnost ogrodja poveča z vtičniki (angl. plugin). Za razvoj vtičnikov je potrebno dobro poznavanje platforme in domorodnega programskega jezika. Če želimo uporabljati vtičnike na različnih platformah, jih je potrebno napisati za vsako platformo posebej.

Potrebno je še omeniti, da vsi API-ji ne delujejo na vseh napravah. Če na primer naprava nima kamere, njenega API-ja ni mogoče uporabiti.

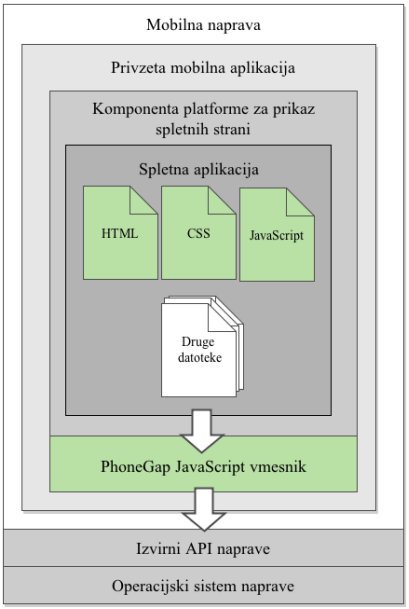
3.1.1 Delovanje ogrodja

Razvijalec s spletnimi tehnologijami razvije spletno aplikacijo, namenjeno mobilnim napravam. Ogrodje s svojimi orodji to aplikacijo pretvori v domorodno aplikacijo želene mobilne platforme.

Ko se aplikacija zažene, se prva spletna stran (običajno index.html) spletne aplikacije naloži v *web view*. Web view je domorodna komponenta platforme, ki se jo uporablja za upodabljanje (angl. render) spletnih strani. To je programsko dostopen ovoj (angl. wrapper) okoli izvirnega spletnega brskalnika platforme. Hkrati z nalaganjem prve spletne strani aplikacije se preda nadzor web viewu. S tem se omogoči uporabnikovo interakcijo s spletno aplikacijo.

Izgled aplikacije je določen s spletnimi stranmi in njihovimi stilskimi predlogami. Logika aplikacije je narejena s skriptnimi jeziki, kot je JavaScript.

Spletne aplikacije nimajo dostopa do različnih komponent naprave, zato ogrodje zagotavlja skupino API-jev, preko katerih je omogočen dostop do določenih komponent naprave. Tako postane spletna aplikacija skoraj enaka domorodnim aplikacijam. Ko aplikacija uporabi klice API-jev, se ti v posebni plasti znotraj aplikacije pretvorijo v ustrezne klice domorodnega API-ja. Slika 3.2 prikazuje delovanje spletnih aplikacij v celoti.



Slika 3.2: Delovanje spletne aplikacije z ogrodjem PhoneGap [15].

3.2 JavaScript

JavaScript je spletni programski jezik, ki so ga razvili v Netscape-u leta 1995. Definiran je v standardu ECMA-262, trenutno je v različici 5.1 in ga podpirajo vsi sodobni spletni brskalniki.

Prvotni namen JavaScripta je bilo potrjevanje oziroma preverjanje spletnih obrazcev, ki so jih do takrat morali preverjati programski jeziki na strežnikovi strani, kot npr. Perl. Od takrat je JavaScript postal mnogo več kot le jezik za pregledovanje pravilnosti spletnih obrazcev. Danes se ga uporablja praktično za vse, kar je povezano z brskalnikovim oknom in njegovo vsebino. Postal je tako pomemben, da so podporo zanj vključili tudi v spletne brskalnike na mobilnih platformah [16]. Prednosti uporabe JavaScripta:

- spletne strani zagotavljajo takojšen odziv na dogodke, ki jih sproži uporabnik, npr. klik na povezavo, vnos v vnosno polje, premik miške...,
- omogoča izdelavo zabavnih, dinamičnih in interaktivnih vmesnikov,

- z njim lahko napišemo praktično vse, od preprostih programov, kot so pojavna okna (angl. pop-up) v spletnem brskalniku, do kompleksnih spletnih aplikacij kot je Google Docs.

Zaradi vse večje popularnosti so se začele množično pojavljati različne JavaScript knjižnice. JavaScript knjižnica je zbirka JavaScript kode, ki ponuja preproste rešitve za probleme, s katerimi se JavaScript razvijalci vsakodnevno srečujejo. Med najbolj znane JavaScript knjižnice sodita jQuery in jQuery Mobile.

3.2.1 jQuery

jQuery je najbolj razširjena JavaScript knjižnica. Ustvarjena je bila z namenom poenostavitve programiranja z JavaScriptom. Z njim je mogoče kompleksno nalogo, ki zahteva dobro poznavanje JavaScripta, rešiti z le nekaj vrsticami kode. Poenostavlja manipulacijo s HTML elementi, z dogodki, animacijami, Ajaxom in mnogimi drugimi stvarmi [9]. Poleg lažjega programiranja poskrbi tudi za konsistentno obnašanje svojih funkcij v vseh spletnih brskalnikih, ki ga podpirajo. Prednosti uporabe knjižnice jQuery:

- majhna velikost knjižnice,
- preprostost uporabe,
- preizkušenost in zanesljivost,
- prosto dostopnost,
- velika razvijalska skupnost,
- mnogo vtičnikov.

3.2.2 jQueryMobile

jQuery Mobile je tudi JavaScript knjižnica [10]. Za razliko od jQueryja se jQuery Mobile osredotoča na mobilne naprave in uporabniške vmesnike.

jQuery Mobile spremeni strukturo HTML in doda svoje stilske predloge tako, da postanejo spletne strani mobilnim napravam bolj prijazne [1]. To pomeni, da se elementi spremenijo tako, da je njihova uporaba na mobilnih napravah lažja. Gumbi in vnosna polja se povečajo, slog in velikost pisave se spremenita, spremeni se postavitev elementov... Najboljšo podporo ponuja mobilnim napravam z operacijskim sistemom Android in iOS.

3.2.3 JSON

JSON je preprost podatkovni format za izmenjavo podatkov. Je tekstovni format in je neodvisen od programskih jezikov. Preprost je za branje in pisanje za uporabnike, kakor tudi razčlenjevanje (angl. parse) in generiranje na elektronskih napravah. Temelji na dveh strukturah [3]:

- **zbirka parov ime/vrednost:** v različnih programskih jezikih je struktura realizirana kot objekt, zapis, struktura, slovar, zgoščevalna tabela ali asociativno polje,
- **urejen seznam vrednost:** v različnih programskih jezikih je struktura realizirana kot polje, vektor, seznam ali zaporedje.

3.3 HTML

HTML je označevalni jezik za izdelavo spletnih strani. Z njim sta predstavljeni vsebina in struktura spletne strani. Programi, ki interpretirajo jezik HTML, so spletni brskalniki. Ti ustrezno interpretirajo jezik HTML in prikažejo spletne strani v uporabniku prijazni obliki.

3.4 CSS

CSS je standard za definiranje vizualne predstavitve spletne strani. Glavni namen uporabe CSS-a je ločiti vsebino spletne strani od oblikovanja (angl.

design). Gre za stilske (oblikovne) predloge HTML dokumenta. V predlogah so definirana pravila, ki določajo, kako se bodo elementi spletne strani obnašali in kako bodo prikazani [5].

3.5 Ajax

Ajax je tehnologija, ki omogoča prenašanje podatkov na in iz spletnega strežnika ter deluje v ozadju. Za takšno delovanje skrbi posebna vmesna plast, imenovana Ajax pogon (angl. engine). To je JavaScript objekt, ki se ga uporabi, ko je potreben prenos podatkov. Delovanje v ozadju ima veliko prednosti. Najbolj pomembna je nemotena uporaba spletne strani ali spletne aplikacije med prenašanjem podatkov, ki jih ta potrebuje za delovanje.

3.6 Mobilni spletni brskalniki

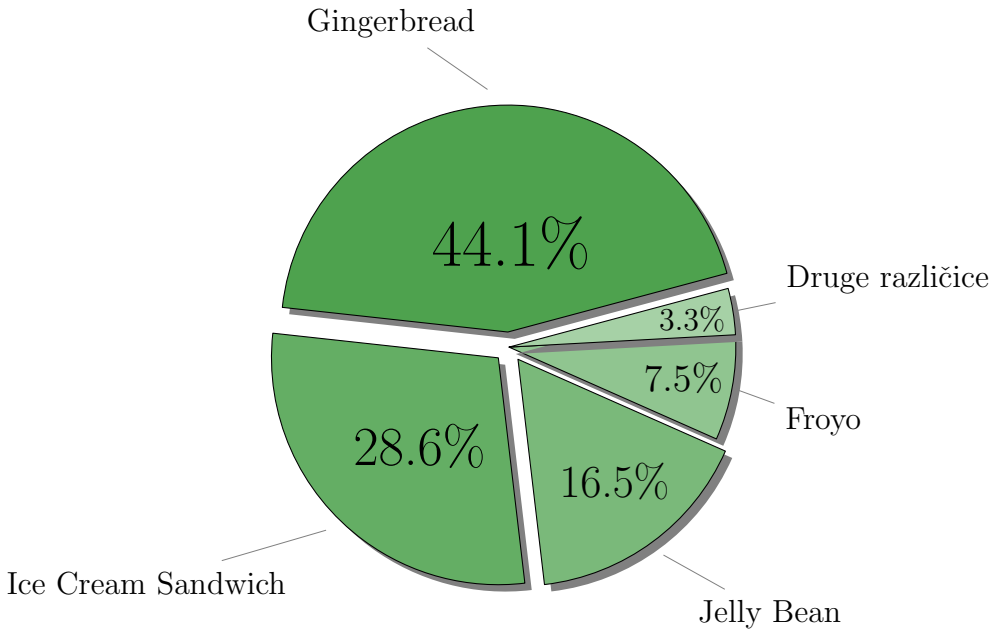
Tako kot za namizne računalnike obstaja tudi za mobilne naprave mnogo različnih spletnih brskalnikov. Praktično vsaka mobilna platforma ima svoj prednameščeni spletni brskalnik. Pri platformi Android je to spletni brskalnik brez posebnega imena. Platforma iOS ima prednameščen brskalnik Mobile Safari, Windows Phone pa Internet Explorer Mobile. Obstajajo še drugi spletni brskalniki, ki pa niso prednameščeni na mobilnih platformah. Med njih spadata Opera Mobile in Mobile Firefox.

Spletni brskalniki Chrome, prednameščeni spletni brskalnik Android, Safari Mobile in v bližnji prihodnosti tudi Opera Mobile uporabljajo enak pogon WebKit [6]. Mobile Firefox uporablja pogon Gecko, Internet Explorer Mobile pa Trident pogon.

3.6.1 Spletni brskalnik Android

Mobilna platforma Android se je prvič pojavila na mobilnih napravah leta 2008. Z vsako novo različico so odpravljene slabosti, ki jih je imela prejšnja, dodane pa so nove funkcionalnosti. Z vsako novo različico se posodobijo tudi

aplikacije. Med njih sodi tudi prednameščeni spletni brskalnik. Tako imamo precej različic spletnih brskalnikov, ki se med seboj razlikujejo v količini podpore standardoma HTML5 in CSS3. To pomeni, da v starejših različicah ni mogoče uporabljati CSS3 animacij ali pa nekaterih API-jev, ki jih ima HTML5. Slika 3.3 prikazuje deleže posameznih različic platforme.



Slika 3.3: Deleži različic platforme Android v marcu 2013 [13].

Poglavje 4

Aplikacija

Namen aplikacije je prenesti reševanje križank s papirja na mobilne naprave. Na voljo sta dva načina reševanja. V prvem načinu igralec rešuje križanke na enak način kot na papirju. Na voljo ima tudi pomoč. Križanke, ki se prenesejo na mobilno napravo, se shranijo v podatkovno bazo. V drugem načinu med seboj tekmujeta dva igralca. Izmenično izpolnjujeta isto križanko in nabirata točke. Vsak lahko vnese naenkrat le eno besedo. Tisti, ki ima na koncu večje število točk, zmaga. Igralcema v tem načinu ni na voljo nobena pomoč.

Pri izdelavi sem uporabil ikone, ki niso plod mojega dela. Te so na voljo na spletni strani [7] in so prosto dostopne. Na sliki 4.1 so prikazane uporabljene ikone.

Poleg ikon sem uporabil še prostodostopna vtičnika *jQuery Finger*, ki ga je mogoče prenesti iz spleta [8], in *spin.js*, ki ga je prav tako mogoče prenesti iz spleta [14]. Prvega sem uporabil za detekcijo dvojnega pritiska, ki služi povečanju in pomanjšanju križanke. Drugega pa za prikaz nalagalnega

Slika 4.1: Uporabljene ikone, ki so ustrezno spremenjene, da ustrezajo izbrani barvni kombinaciji aplikacije.

zaslona. Ta se prikaže med nalaganjem in prenašanjem podatkov.

Aplikacija je sestavljena iz štirih delov: domorodnega dela, strukture HTML, stilskih predlog in logike. Domorodni del je osnova aplikacije in je napisan v programskem jeziku Java. Struktura HTML definira strukturo aplikacije in skupaj s stilskimi predlogami skrbi za njen izgled. Logika je napisana v programskem jeziku JavaScript.

4.1 Domorodni del aplikacije

Vsaka aplikacija, razvita z ogrodjem PhoneGap, potrebuje domorodni del. V njem sem ustvaril razred (angl. class), ki razširja aktivnost *DroidGap*. Ta nadomešča klasično aktivnost platforme Android. To je posebna aktivnost, ki lahko prikazuje spletne strani. V njej sem določil začetni zaslon (angl. splash screen) in začetno stran aplikacije, ki se prikaže, ko se aplikacija naloži. Poleg tega sem v aktivnosti odstranil navpični drsnik (angl. scroll bar). Aktivnost je prikazana v programski kodi 4.1.

```
1 public class KrizankaActivity extends DroidGap{
2  @Override
3  public void onCreate(Bundle savedInstanceState){
4   super.onCreate(savedInstanceState);
5   super.setIntegerProperty("splashscreen",R.drawable.splashscreen);
6   super.loadUrl("file:///android_asset/www/index.html",60000);
7   super.appView.setVerticalScrollBarEnabled(false);
8  }
9 }
```

Programska koda 4.1: Domorodni del aplikacije. Razred *KrizankaActivity* razširja posebno aktivnost *DroidGap*. Z metodo *setIntegerProperty* sem nastavil začetni zaslon, z metodo *loadUrl* pa začetno stran aplikacije.

V domorodni del aplikacije sodi tudi manifest. V njem sem določil nekatere nastavitve in lastnosti aplikacije. Določil sem podporo različnim velikostim zaslonov, orientacijo zaslona (pokončno), temo (okno brez naslovne vrstice (angl. title bar)), obnašanje programske tipkovnice, dovoljenja, ki jih

aplikacija potrebuje za delovanje (dostop do omrežja in dovoljenja za shranjevanje podatkov), in minimalno zahtevano različico platforme, na kateri aplikacija še deluje. Z obnašanjem programske tipkovnice sem določil, da se pri prikazu tipkovnice okno aplikacije ne prekrije s tipkovnico, temveč se pomanjša.

4.2 HTML struktura aplikacije

Struktura in nekaj vsebine aplikacije je določena s HTML-jem. Celotno strukturo aplikacije predstavlja ena HTML datoteka. Ta je sestavljena iz dveh delov: glave (angl. head) in telesa (angl. body).

V glavi, ki je prikazana v programski kodi 4.2, sem določil lastnosti dokumenta HTML. V znački *meta*, z imenom*viewport*, sem določil višino in širino okna aplikacije ter onemogočil možnost spreminjanja njegove velikosti. V naslednji znački sem določil vrsto vsebine dokumenta in vrsto kodiranja znakov. V značkah *link* in *script* sem navedel poti do datotek s stilskimi predlogami in kodo JavaScript. Izjema je zadnja značka script. V njej sem povezal dokument s PhoneGap dogodkom *deviceready*. To je dogodek, ki se sproži, ko se PhoneGap naloži in je pripravljen na izvajanje. Ko pride do tega dogodka, se pokliče funkcijo *onDeviceReady*, ki je začetna funkcija aplikacije.

Telo predstavlja vsebino aplikacije. To je sestavljeno iz klasičnih značk HTML in jQuery Mobile gradnikov (angl. widget). Gradniki so določeni s podatkovnimi atributi (angl. data attributes). V telesu sem ustvaril več strani. Vsaka stran predstavlja zaslon aplikacije. Vsaka stran je gradnik, ki je določen znotraj div značke HTML z dodanim podatkovnim atributom *data-role="page"*. Vsaki od teh strani sem določil, kakšno zgradbo bo imela in katere gradnike bo vsebovala. Vsa pojavna okna (angl. pop-up) so definirana znotraj strani, na kateri se bo to pojavno okno pojavljalo. To zahteva jQuery Mobile enako kot določitev več strani znotraj telesa dokumenta HTML. Pojavna okna sem določil z podatkovnim atributom *data-role="popup"*. Programska koda 4.3 prikazuje primer strani z glavnim menijem in s pojavnim

oknom.

```
<head>
 <meta name="viewport" content="width=device-width, height=device-height,
      initial-scale=1, user-scalable=0">
 <meta http-equiv="content-type" content="text/html; charset=utf-8" />
 <link rel="stylesheet" type="text/css" href="css/jquery.mobile-1.3.0.css" />
 <link rel="stylesheet" type="text/css" href="css/CrosswordStyle.css" />
 <script src="cordova-2.7.0.js" type="text/javascript"></script>
 <script src="js/jquery-v1.9.1.js" type="text/javascript"></script>
 <script src="js/jquery.mobile-1.3.0.js" type="text/javascript"></script>
 <script src="js/jquery.finger.js" type="text/javascript"></script>
 <script src="js/spin.js" type="text/javascript"></script>
 <script src="js/CrosswordScript.js" type="text/javascript"></script>
 <script>
  $(document).bind('deviceready',onDeviceReady);
 </script>
</head>
```

Programska koda 4.2: Vsebina značke glava. V značkah *meta* je določena velikost zaslona aplikacije in kodiranje znakov. V značkah *link* in *script* so navedene poti do datotek s stilskimi predlogami in JavaScript kodo. Zadnja značka script poveže dokument s PhoneGap dogodkom *deviceready*. Ob sprožitvi dogodka se pokliče začetno funkcijo aplikacije.

Vsaka stran vsebuje gradnike jQuery Mobile. Nekatere vsebujejo samo gumbe z ikonami, druge navigacijo z zavihki (angl. tab), sezname za prikaz podatkov, panelne menije itd. Vse gradnike sem določil s podatkovnimi atributi. Navigacijo z zavihki vsebujeta menija v obeh načinih igranja. Določena je s podatkovnim atributom *data-role="navbar"*. Za prikaz uporabnikovih križank, iger, prijateljev in igralcev sem uporabil sezname. Določeni so s podatkovnim atributom *data-role="listview"*. Vsebina se doda seznamom dinamično med izvajanjem aplikacije. Seznamoma prijateljev in igralcev sem dodal tudi filter. Ta omogoča lažje iskanje med večjimi količinami podatkov. Določil sem ga s podatkovnim atributom *data-filter="true"*. Panelna menija, ki se ju uporablja med reševanjem križank, sem določil s podatkovnim atributom *data-role="panel"*. Namenjena sta prikazu gumbov, rezultata in

pomoči.

```
<div id="MainPage" data-role="page" data-theme="a">
  <div data-role="navbar" class="stretch top-fixed">
    <ul>
        <li><a href="#" data-role="tab" class="ui-state-active">Select game</a></li>
      </ul>
  </div>
  <div data-role="content" class="content">
    <ul data-role="listview" data-inset="true" data-icon="false" >
      <li>
        <a id="SinglePlayer" href="#SinglePlayerPage">
          <div class="list-icon icon-singleplayer"></div>
          <h2>Single player</h2>
          <p>Solve crosswords alone</p>
        </a>
      </li>
      <li>
        <a id="MultiPlayer" href="#">
          <div class="list-icon icon-multiplayer"></div>
          <h2>Multiplayer</h2>
          <p>Compete with your friends</p>
        </a>
      </li>
      <li>
        <a id="Logout" href="#LoginPage">
          <div class="list-icon icon-logout"></div>
          <h2>Log out</h2>
          <p>Log out from application</p>
        </a>
      </li>
    </ul>
  </div>
  <div id="MainPopup" data-role="popup" data-overlay-theme="a" data-dismissible="false" data-add-back-btn="true">
    <div data-role="navbar">
      <ul>
          <li><a id="MainPopupTitle" href="#" data-role="tab" class="ui-state-active">Težave s povezavo</a></li>
      </ul>
    </div>
    <div data-role="content">
      <div class="popup-content">
        <h3 id="MainPopupSubtitle" class="ui-title gray-text"></h3>
        <p id="MainPopupText">Za igranje v večigralskem načinu mora biti vključen Wi-Fi ali paketni prenos podatkov.</p>
        <div id="MainPopupIcon" class="img-icon icon-wifi"></div>
```

```
43          </div>
44          <a id="MainPopupClose" href="#" data-role="button" data-theme="a">V
                redu</a>
45        </div>
46      </div>
47  </div>
```

Programska koda 4.3: Primer strani oziroma zaslona s pojavnim oknom. To je stran z glavnim menijem, kjer je možno izbrati način igranja ali pa se odjaviti iz aplikacije. Pojavno okno je določeno s podatkovnim atributom *data-role="popup"*.

4.3 Shranjevanje podatkov

PhoneGap ponuja dva načina shranjevanja podatkov. Prvi je lokalno shranjevanje podatkov v obliki parov ključ-vrednost (angl. key-value), drugi pa shranjevanje v lokalno podatkovno bazo. V aplikaciji sem uporabil oba načina, saj ima vsak svojo prednost.

4.3.1 Lokalno shranjevanje parov ključ-vrednost

Lokalno shranjevanje podatkov v obliki parov ključ-vrednost je preprosto in dolgotrajno. To pomeni, da se podatki, shranjeni na ta način, ohranijo tudi po izklopu aplikacije. Ta način shranjevanja sem uporabil za shranjevanje dveh nastavitev aplikacije.

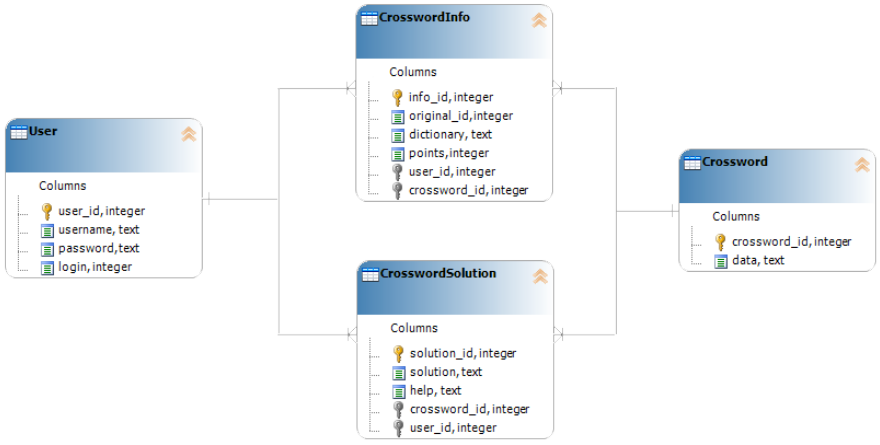
Prva nastavitev je višina programske tipkovnice, druga pa uporabniško ime zadnjega prijavljenega uporabnika. Na ta način sem realiziral avtomatsko prijavljanje v aplikacijo. To pomeni, da se v enoigralskem načinu samodejno ustvari seznam uporabnikovih križank, v večigralskem načinu pa seznam uporabnikovih iger, ki se prenese s strežnika. Branje in shranjevanje lokalno shranjenih podatkov je prikazano v programski kodi 4.4.

```
1 var keyboard = window.localStorage.key("keyboard");
2 var keyboardHeight = CONFIG.AppHeight - $(window).height();
3 window.localStorage.setItem("keyboard", keyboardHeight);
```

Programska koda 4.4: Uporaba shranjevanja parov ključ-vrednost. V prvi vrstici se prebere vrednost, ki je shranjena pod ključem *keyboard*. V zadnji vrstici se shrani vrednost *keyboardHeight* pod ključem *keyboard*.

4.3.2 Podatkovna baza

Podatkovno bazo sem uporabil za shranjevanje uporabnikov, prenesenih križank in dodatnih podatkov o njih ter uporabnikovih vnosov. Podatkovno bazo, katere struktura je prikazana na sliki 4.2, sestavljajo štiri tabele: User, CrosswordInfo, Crossword in CrosswordSolution.



Slika 4.2: Struktura podatkovne baze, ki je sestavljena iz štirih tabel.

Tabela User

Tabela vsebuje osnovne podatke o uporabnikih aplikacije. Vsebuje naslednje štiri stolpce:

- **user_id:** primarni ključ,
- **username:** uporabniško ime,
- **password:** uporabnikovo geslo,
- **login:** stanje uporabnika (prijavljen/neprijavljen).

Tabela Crossword

Tabela vsebuje podatke o križankah, ki jih uporabnik prenese s strežnika na mobilno napravo. Vsebuje naslednja dva stolpca:

- **crossword_id:** primarni ključ,
- **data:** križanka, shranjena kot niz.

Tabela CrosswordInfo

Tabela vsebuje nekaj splošnih podatkov o križanki in služi predvsem kreiranju seznama križank. Vsebuje naslednjih 6 stolpcev:

- **info_id:** primarni ključ,
- **original_id:** originalni id križanke, enak tistemu na strežniku,
- **dictionary:** jezik križanke,
- **points:** število točk, ki jih doseže uporabnik,
- **user_id:** tuji ključ tabele User,
- **crossword_id:** tuji ključ tabele Crossword.

Tabela CrosswordSolution

Tabela vsebuje podatke o uporabnikovih vnosih v križanko in uporabljeni pomoči. Vsebuje naslednjih pet stolpcev:

- **solution_id:** primarni ključ,
- **solution:** uporabnikova rešitev, shranjena kot niz,
- **help:** uporabnikova pomoč, shranjena kot niz,
- **crossword_id:** tuji ključ tabele Crossword,
- **user_id:** tuji ključ tabele User.

Kreiranje podatkovne baze

Ob vsakem zagonu aplikacije se najprej preveri, če podatkovna baza že obstaja. Če ne obstaja, se jo ustvari, drugače pa se jo le odpre. Programska koda 4.5 in 4.6 prikazuje kreiranje podatkovne baze in nove tabele v njej. Vsaki tabeli sem določil imena stolpcev in tipe vrednosti, ki jih vsebujejo.

```
1 init: function(){
2  DATABASE.DB = window.openDatabase("crosswordsDB","1.0","Crosswords"
       ,10*1024*1024);
3  DATABASE.createTableUser();
4  DATABASE.createTableCrosswordInfo();
5  DATABASE.createTableCrossword();
6  DATABASE.createTableCrosswordSolution();
7 }
```

Programska koda 4.5: Kreiranje podatkovne baze. V prvi vrstici se ustvari prazna podatkovna baza. Podatkovni bazi se določi ime, s katerim bo shranjena, verzijo, prikazano ime in velikost. Preostala koda ustvari potrebne tabele v podatkovni bazi.

```
1 createTableCrossword: function(){
2  DATABASE.DB.transaction(
3   function(tx){
4    tx.executeSql('CREATE TABLE IF NOT EXISTS Crossword (crossword_id INTEGER
         PRIMARY KEY UNIQUE, data TEXT)');
5   },
6   function(err){
7    console.log("DATABASE ERROR, function createTableCrossword, error: "+err.
       code+" , message: "+err.message);
8  });
9 }
```

Programska koda 4.6: Kreiranje tabele *Crossword*. Tabela je sestavljena iz dveh vrstic. Prva je primarni ključ, druga pa je križanka. Križanka se pred shranjevanjem v tabelo pretvori iz formata JSON v niz (angl. string).

4.4 Uporabniški vmesnik

Aplikacija je sestavljena iz petih zaslonov:

- **prijava:** prijava uporabnika v aplikacijo,
- **glavni meni:** izbira načina igranja in odjava iz aplikacije,
- **enoigralski meni:** izbira in prenos novih križank,
- **večigralski meni:** izbira in kreiranje igre ter pregled lestvic,
- **igra:** reševanje križank.

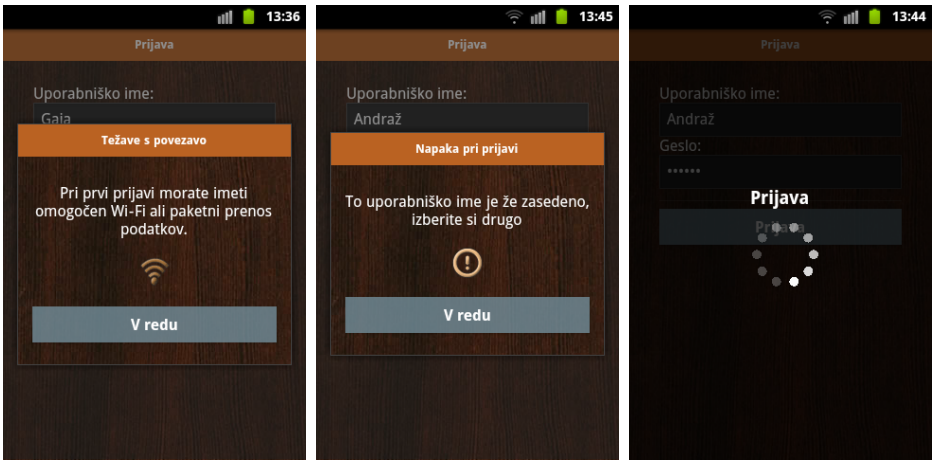
Vsak zaslon predstavlja stran jQuery Mobile v dokumentu HTML. Med zasloni aplikacije se premika z ukazom *$.mobile.changePage('#ID_strani')*. ID_strani je unikatni identifikator strani, definirane v dokumentu HTML.

4.4.1 Prijava

Aplikacijo sem zasnoval tako, da se mora vsak uporabnik prijaviti. Prijavi se z uporabniškim imenom in geslom. To je potrebno storiti ob prvem zagonu aplikacije. Uporabnik mora vnesti poljubno uporabniško ime in svoje geslo. Ta podatka se pošljeta na strežnik, kjer se preveri, če je želeno uporabniško ime prosto. Če je, se kreira nov uporabnik na strežniku in v podatkovni bazi naprave. Če uporabniško ime ni več prosto, se prikaže pojavno okno z ustreznim obvestilom. V tem primeru se ne ustvari nov uporabnik. Na napravi mora biti vključen Wi-Fi ali pa paketni prenos podatkov (v nadaljevanju samo prenos podatkov). Če ni, se prikaže ustrezno obvestilo. Na slikah 4.3 in 4.4 so prikazani prijavni zaslon in obvestila.

Prijavni obrazec sem naredil tako, da je na začetku mogoč vnos le uporabniškega imena (vnosnemu polju za geslo sem dodal atribut onemogočeno (angl. disabled)). Na ta način sem poskrbel, da brskalnik ne zazna dveh vnosnih polj. S tem sem preprečil prikaz dodatne navigacijske vrstice nad tipkovnico (gumba nazaj in naprej). Ta se pojavi kot pomoč za lažje premikanje po obrazcu. Ko uporabnik vnese uporabniško ime, se omogoči vnos

Slika 4.3: Prvi sliki prikazujeta prijavni obrazec, tretja pa opozorilo za nepopolno izpolnjen obrazec.



Slika 4.4: Prva slika prikazuje obvestilo o nevključenosti prenosa podatkov, druga obvestilo o zasedenosti uporabniškega imena in tretja nalagalni zaslon, ki se prikaže zaradi čakanja na strežnikov odgovor.

gesla (odstrani se atribut onemogočeno). Nato uporabnik izbere vnosno polje za vnos gesla. Takrat se sproži dogodek, v katerem se izračuna velikost tipkovnice. Ta je enaka razliki med velikostjo celotnega zaslona in trenutno velikostjo okna aplikacije, ki se je spremenila zaradi prikaza tipkovnice. Tako sem dobil velikost tipkovnice, ki jo v sami igri uporabljam za premike križanke, ko je tipkovnica prikazana. Ta podatek se lokalno shrani, da je na voljo vsakič, ko se aplikacija zažene. Tipkovnica z dodatno vrstico je prikazana na sliki 4.3.

Ko uporabnik vnese zahtevana podatka, mora pritisniti na gumb Prijava. Takrat se oba vnešena podatka pošljeta z zahtevo Ajax na strežnik. V programski kodi 4.7 so prikazani zahteva za strežnik in povratna klica (angl. callback), ki se izvedeta ob uspešno ali neuspešno izvršeni zahtevi. V primeru uspešno izvršene zahteve se ustvari nov uporabnik v lokalni podatkovni bazi. Takrat se prikaže zaslon z glavnim izbirnim menijem. V primeru neuspešno izvedene zahteve se preveri strežnikov odgovor. Če ta sporoča, da uporabnik že obstaja, se prikaže pojavno okno z ustreznim obvestilom. Ob drugih napakah se prikaže pojavno okno z drugim obvestilom.

```
createPlayer: function(username,password){
 LOCK_SCREEN.show("Prijava", "");
 var data = {
  "Username": username,
  "Password": password,
 };
 $.ajax({
  type:'POST',
  url:'http://crosscompete.com/api/Players/LogIn',
  dataType:'json',
  contentType:'application/json',
  data: JSON.stringify(data),
  }).done(function(result){
   DATABASE.createUser(username, password);
   LOCK_SCREEN.hide();
  }).fail(function(result){
   LOCK_SCREEN.hide();
   var message=JSON.parse(result.responseText);
   if (message.Message === "Name is already taken"){
    var  title="Napaka pri prijavi",
         text="To uporabniško ime je že zasedeno, izberite si drugo",
```

```
22            icon="icon-error";
23        MENU.createLoginPopup(title,text,icon);
24        MENU.showLoginPopup();
25       }else{
26        var title="Napaka na strežniku",
27            text="Prišlo je do napake na strežniku. Poiskusite se ponovno
                  prijaviti",
28            icon="icon-error";
29        MENU.createLoginPopup(title,text,icon);
30        MENU.showLoginPopup();
31       }
32    });
33 }
```

Programska koda 4.7: Pošiljanje zahteve Ajax za prijavo novega uporabnika. Če se zahteva uspešno izvede, se izvrši povratni klic *končano* (angl. done), drugače pa povratni klic *neuspeh* (angl. fail).

Ob uspešno izvedeni zahtevi se doda nov uporabnik v lokalno podatkovno bazo. Doda se ga v tabelo *User*. Vsakemu uporabniku sem, poleg uporabniškega imena in gesla, dodal še vrstico *login*. Ta ima po prijavi vrednost 0. To pomeni, da je uporabnik prijavljen v aplikacijo. Po uspešnem kreiranju uporabnika se lokalno shrani tudi njegovo uporabniško ime, enako kot podatek o višini tipkovnice. Ko se aplikacija naslednjič zažene, se preveri, če je kakšen uporabnik že prijavljen. Če je, se prikaže zaslon z glavnim izbirnim menijem, drugače pa zaslon s prijavo. Ko se uporabnik odjavi iz aplikacije, se ponastavita vrednost login v podatkovni bazi in vrednost, ki je shranjena lokalno.

4.4.2 Glavni meni

Ko se aplikacija začne izvajati, se preveri, če je kakšen uporabnik že prijavljen v aplikacijo. Če ni nobenega prijavljenega uporabnika, se prikaže zaslon za prijavo. V nasprotnem primeru se prenese iz podatkovne baze uporabnikov id (id, kot ga ima v podatkovni bazi), nato pa se prikaže zaslon z glavnim menijem. Prikaz zaslona z glavnim menijem sem zakasnil za dve sekundi. To

sem naredil zaradi prikaza strani za prijavo pred stranjo z glavnim menijem. Ker je v strukturi HTML stran za prijavo ustvarjena pred stranjo z glavnim menijem, se v vsakem primeru prikaže pred njo. To sem rešil z daljšim prikazom pozdravnega zaslona. Med tem časom pride do zamenjave aktivne strani v ozadju, zato prijavne strani ne vidimo. V programski kodi 4.8 je prikazana izbira prve strani.

```
1  login: function(){
2   var keyboard=window.localStorage.getItem("keyboard"),
3            user=window.localStorage.getItem("user");
4   if ((user === null)||(user === 'null')){
5    $.mobile.changePage('#LoginPage');
6   }else{
7    $.mobile.changePage('#MainPage');
8    DATABASE.getUser(user);
9   }
10  setTimeout(function(){navigator.splashscreen.hide()},2000);
11 }
```

Programska koda 4.8: Izbira začetnega zaslona. Preveri se, če obstaja lokalno shranjen podatek o aktivnem uporabniku. Če tega podatka ni, se prikaže zaslon za prijavo, drugače pa se prikaže zaslon z glavnim menijem. Z zadnjo vrstico se za dve sekundi zakasni odstranitev pozdravnega zaslona.

V glavnem menju lahko uporabnik izbira med enoigralskim in večigralskim načinom igranja ter odjavo iz aplikacije.

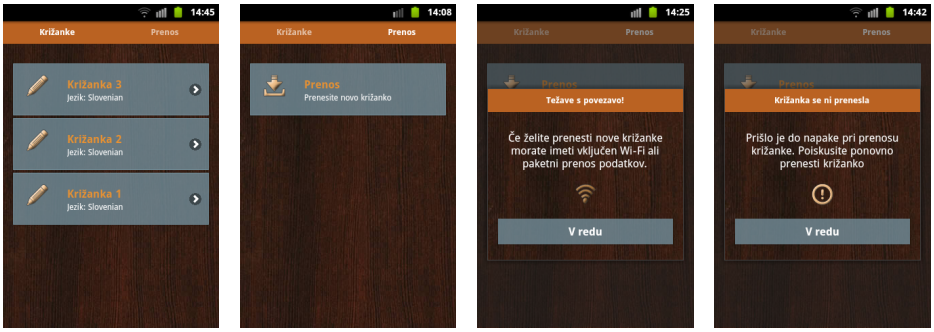
Če uporabnik izbere večigralski način in nima vključenega prenosa podatkov, se prikaže pojavno okno z obvestilom. Meni in obvestilo sta prikazana na sliki 4.5.

Ob odjavi iz aplikacije se vrstica login trenutnega uporabnika v bazi spremeni na 1. To pomeni da se je odjavil. Poleg tega se izbriše tudi lokalno shranjen podatek o trenutnem uporabniku.

Slika 4.5: Na prvi sliki je prikazan glavni meni, na drugi pa opozorilo za vklop prenosa podatkov.

4.4.3 Enoigralski meni

Enoigralski meni je sestavljen iz dveh delov. Prvi del je namenjen prikazu uporabnikovih križank, ki so shranjenje v podatkovni bazi. Drugi del je namenjen prenosu križank s strežnika. Oba sta prikazana na sliki 4.6.



Slika 4.6: Na prvi sliki je prikazan seznam uporabnikovih križank, na drugi prenos križank, na tretji pojavno okno za vklop prenosa podatkov in na zadnji pojavno okno z obvestilom o napaki pri prenosu križanke.

Ko uporabnik izbere enoigralski način, se iz podatkovne baze prenesejo podatki o njegovih križankah. Ti se nahajajo v tabeli *CrosswordInfo*. Iz teh

podatkov se ustvari seznam križank, ki se prikaže na prvem zavihku enoigralskega načina. Na drugem zavihku je gumb za prenos križank s spletnega strežnika.

Seznam križank sem zgradili tako, da sem v strukturi HTML pripravil seznam. Ko se vsi podatki o križankah prenesejo iz podatkovne baze, se za vsako križanko ustvari značka HTML (*li*), ki predstavlja element seznama. V vsako značko se doda razrede za stilske predloge, ikone, id, ki je enak id-ju, ki ga ima križanka v podatkovni bazi, in id vrstice v tabeli CrosswordInfo. Doda se tudi besedilo, ki predstavlja številko križanke in jezik. Vse ustvarjene značke li se dodajo prej pripravljenemu seznamu. Ko so dodani vsi elementi, se seznam osveži. Tako se prikažejo novi podatki. Na koncu se ustvarijo še poslušalci dogodkov (angl. event listener). Vsaki znački li sem dodal razred *single-player-crossword*. Na ta razred sem povezal poslušalca. Ko uporabnik izbere eno križanko s seznama, se sproži dogodek. Nato je potrebno ugotoviti, kateri element je bil pritisnjen. To sem naredil s podatkom id, ki ga vsebuje pritisnjeni element. Id je enak id-ju križanke v podatkovni bazi. Ta podatek se uporabi za prenos križanke iz podatkovne baze. V programski kodi 4.9 je prikazano kreiranje seznama.

```
1  createSinglePlayerCrosswordList: function(queryResult){
2    $('#SinglePlayerCrosswordsList').empty();
3    for (var i=0;i<queryResult.rows.length;i++)
4    {
5      $('<li data-icon="arrow-r"><a href="#" id="'+queryResult.rows.item(i).
         crossword_id+'" class="single-player-crossword"><div class="list-
         icon icon-single-crossword"></div><div class="list-item"><h2>
         Crossword '+queryResult.rows.item(i).original_id+'</h2><p>Language:
          '+ queryResult.rows.item(i).dictionary+'</p><p>Points: '+
         queryResult.rows.item(i).points+'</p></div></a></li>').appendTo($('#
         SinglePlayerCrosswordsList'));
6      $('#'+queryResult.rows.item(i).crossword_id).data('info',{infoDBID:
         queryResult.rows.item(i).info_id});
7    }
8    $('#SinglePlayerCrosswordsList').listview('refresh');
9    $('.single-player-crossword').unbind().bind('click',function(){
10     LOCK_SCREEN.show("","");
11     APP.crossword.crosswordDBID=$(this).attr('id');
12     var info=$(this).data('info');
13     APP.crossword.infoDBID=info.infoDBID;
```

```
14      DATABASE.getCrossword($(this).attr('id'));
15    });
16  }
```

Programska koda 4.9: Kreiranje seznama uporabnikovih križank. V prvem delu se ustvari značke *li*, ki predstavljajo elemente seznama. Te značke se doda seznamu. V drugem delu se doda poslušalca dogodkov za vsak element seznama, da se lahko zazna izbiro križanke.

Prenos križank s strežnika v podatkovno bazo se izvede, ko uporabnik pritisne gumb za prenos. Ob pritisku na gumb se iz podatkovne baze prebere id zadnje dodane križanke. Z zahtevo Ajax se ta podatek pošlje na strežnik. Strežnik vrne odgovor z novo križanko, ki ima id za ena večji od poslanega. Prejeta križanka se shrani v podatkovno bazo. Če pride do napake med prenosom, se prikaže pojavno okno z obvestilom. V programski kodi 4.10 je prikazan prenos križanke.

Ko se križanka shrani, se naredi vnos v dve tabeli. V tabelo *Crossword* se shrani prejeta križanka. Ta se predhodno iz formata JSON pretvori v niz. To sem naredil z ukazom *JSON.stringify(crossword)*. Drugi vnos se naredi v tabelo *CrosswordInfo*. Vanjo se dodajo splošni podatki o križanki, tuji ključ pravkar dodane križanke in tuji ključ tabele uporabnika. Ključa se dodata za ustrezno povezanost tabel.

```
1  downloadCrossword: function(crossword_id){
2   $.ajax({
3    type:'GET',
4    url:'http://crosscompete.com/api/Crosswords/Get/'+crossword_id,
5    dataType:'json'
6    }).done(function(result){
7     if (result.Erorr === undefined){
8      DATABASE.saveCrossword(result);
9     }else{
10     LOCK_SCREEN.hide();
11     var title="Križanka se ni prenesla",
12          text="Prišlo je do napake pri prenosu križanke. Poiskusite ponovno
                prenesti križanko",
13          icon="icon-error";
14     MENU.createSinglePlayerPopup(title,text,icon);
15     MENU.showSinglePlayerPopup();
```

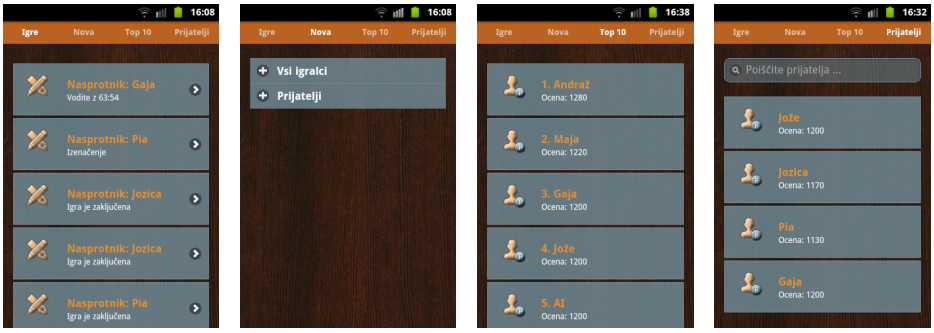
```
16      }
17    }).fail(function(){
18      LOCK_SCREEN.hide();
19      var title="Križanka se ni prenesla",
20          text="Prišlo je do napake pri prenosu križanke. Poiskusite ponovno
                prenesti križanko",
21          icon="icon-error";
22      MENU.createSinglePlayerPopup(title,text,icon);
23      MENU.showSinglePlayerPopup();
24    });
25  }
```

Programska koda 4.10: Pošiljanje zahteve Ajax za prenos križanke. Če je bila zahteva uspešno zaključena, se izvede povratni klic *končano*, drugače pa povratni klic *neuspeh*. Čeprav je bila zahteva uspešno izvedena, se preveri, če je slučajno prišlo do kakšne druge napake. Če je ni bilo, se križanka shrani v podatkovno bazo.

4.4.4 Večigralski meni

Meni je sestavljen iz štirih delov: seznama uporabnikovih iger, ustvarjanja novih iger, seznama najboljših desetih igralcev in seznama prijateljev. Prikazani so na sliki 4.7. Ko uporabnik v glavnem meniju izbere večigralski način igranja, se na strežnik pošljejo štiri zahteve Ajax, ki zahtevajo prenos:

- seznama uporabnikovih iger,
- seznama desetih najboljših igralcev,
- seznama uporabnikovih prijateljev,
- seznama vseh igralcev.



Slika 4.7: Prva slika prikazuje seznam uporabnikovih iger, druga slika kreiranje nove igre, tretja slika seznam desetih najboljših igralcev in zadnja slika seznam prijateljev.

Uporabnika se preusmeri v večigralski meni šele, ko se izvedejo vse štiri zahteve. Med tem je prikazan nalagalni zaslon, ki opozarja uporabnika na prenos podatkov. Preusmeritev v meni, po izvršitvi vseh zahtev, sem dosegel z uporabo jQuery objekta zakasnitve (angl. deferred). Ta omogoča, da se programska koda nadaljuje z izvajanjem šele, ko se izvedejo vse poslane zahteve. V programski kodi 4.11 je prikazana uporaba objekta zakasnitve. Ko se izvedejo vse zahteve, se iz prenesenih podatkov ustvarijo seznami.

Seznama najboljših desetih igralcev in prijateljev sta namenjena samo branju. Seznam prijateljev ima dodan tudi filter. Ta koristi, če ima igralec veliko prijateljev.

```
1  LOCK_SCREEN.show("Prenašanje podatkov");
2  $.when(
3      SERVER.getUserGames(APP.user.username),
4      SERVER.getTopPlayers(),
5      SERVER.getFriends(APP.user.username),
6      SERVER.getAllPlayers()
7  ).then(
8      function(result1,result2,result3,result4){
9        APP.game.userGames = JSON.parse(result1[2].responseText);
10       APP.game.topPlayers = JSON.parse(result2[2].responseText);
11       APP.game.userFriends = JSON.parse(result3[2].responseText);
12       APP.game.allPlayers = JSON.parse(result4[2].responseText();
13       $.mobile.changePage('#MultiPlayerPage');
14       MENU.createMultiPlayerGamesList(APP.game.userGames);
15       MENU.createTopPlayersList(APP.game.topPlayers);
```

```
16          MENU.createFriendsList(APP.game.userFriends);
17          MENU.createNewGamePlayersList(APP.game.allPlayers);
18          MENU.createNewGameFriendsList(APP.game.userFriends);
19          LOCK_SCREEN.hide();
20          MENU.startTimer();
21      }
22  );
```

Programska koda 4.11: Uporaba objekta zakasnitve. Prenos vseh potrebnih podatkov za kreiranje seznamov v večigralskem načinu s pomočjo objekta zakasnitve. Vsaka poslana zahteva vrne svoj rezultat, ki se ga uporabi za kreiranje seznamov.

Seznama uporabnikovih iger in izbira igralca v naslednji igri sta ustvarjena enako kot seznam uporabnikovih križank, ki je opisan v podpoglavju 4.4.3. Edina sprememba je določitev id-ja. Pri seznamu iger je ta enak id-ju igre na strežniku, pri seznamu za izbiro igralca pa se ga ne določi. Igralec je namreč unikatno določen že z uporabniškim imenom.

Seznam iger vsebuje vse aktivne igre in zadnje tri, ki so že končane. Ko uporabnik izbere igro s seznama, se pošljeta na strežnik dve zahtevi Ajax z uporabo objekta zakasnitve. Prva zahteva pošlje id križanke. Nanjo strežnik odgovori s križanko. Druga zahteva pošlje id igre. Nanjo strežnik odgovori s seznamom potez, ki so že bile storjene.

Če želi uporabnik ustvariti novo igro, si mora najprej izbrati nasprotnika. Tega izbere s seznama vseh igralcev ali pa s seznama svojih prijateljev. Vsak nov igralec, s katerim uporabnik začne igro, se doda na strežniku med prijatelje. Ko izbere nasprotnika, se na strežnik pošlje zahteva Ajax skupaj z izbranim nasprotnikom. Če igralca nimata aktivne igre, se igra ustvari, sicer pa ne. V slednjem primeru se uporabniku, ki je želel ustvariti novo igro, prikaže pojavno okno z obvestilom, da igre ni bilo mogoče ustvariti. V primeru uspešno ustvarjene igre se pošlje še zahteva za prenos križanke. Programska koda 4.12 prikazuje zahtevo Ajax za ustvarjanje nove igre.

V večigralskem meniju sem ustvaril tudi samodejno osveževanje seznamov s pomočjo časovnika (angl. timer). Časovnik se sproži, ko postane aktivna

stran z glavnim menijem. Ko ta stran ni aktivna, časovnik stoji. Časovnik vsako minuto pošlje štiri zahteve Ajax za prenos seznamov, enako kot ob izbiri večigralskega načina.

```
createNewGame: function(user1,user2){
  LOCK_SCREEN.show("Kreiranje nove igre","");
  $.ajax({
    type:'GET',
    url:'http://crosscompete.com/api/Games/Create/'+user1+'/'+user2,
    dataType: 'json'
  }).done(function(result){
    APP.game.gameState = result;
    $.when(
          SERVER.getCrossword( APP.game.gameState.Crossword.ID)
    ).then(
          function(result){
        APP.crossword.data = result;
        CROSSWORD.create([APP.game.gameState.LastMove]);
      }
    );
  }).fail(function(result){
    LOCK_SCREEN.hide();
    var message =(JSON.parse(result.responseText));
    if (message.Message === "You already have a game with this player"){
      var title = "S tem igralcem že igrate igro",
          text = "Z istim igralcem imate lahko hkrati aktivno le eno igro",
              icon = "icon-info";
      MENU.createMultiPlayerPopup(title,text,icon);
      MENU.showMultiPlayerPopup();
    }
  });
}
```

Programska koda 4.12: Pošiljanje zahteve Ajax za kreiranje nove igre. Če je zahteva za kreiranje nove igre uspešna, se pošlje še zahteva za prenos križanke. Če zahteva ni uspešna, se prikaže pojavno okno z obvestilom.

4.4.5 Igra

Glavni del aplikacije je zaslon s prikazom križanke. Zaslon je sestavljen iz prikaza vprašanja, križanke, menija, dveh panelnih menijev in pojavnega

okna. V programski kodi 4.13 je prikazana stran s križanko.

Ko uporabnik pritisne na kvadratek križanke, se izbere beseda. Njeno vprašanje se izpiše v polje nad križanko. Če je vprašanje daljše od ene vrstice, se velikost črk zmanjša, prav tako se zmanjša tudi višina vrstice elementa, v katerega se izpiše vprašanje. To sem naredil s spremembo stilske predloge elementa v JavaScript kodi. Na ta način sem obdržal enako velikost prikazovalnega polja pri različno dolgih vprašanjih.

Osnova križanke je značka div z id-jem *CrosswordContainer*. Ta značka je vsebovalnik križanke. Njej je dodana značka div z id-jem *Crossword*. Ta značka predstavlja križanko. Njej se dodajo vrstice križanke, njim pa posamezni kvadratki. V vsebovalnik križanke sem dodal tudi vnosno polje, ki se ga uporablja za vnos črk. To polje je skrito. Več o izgradnji in delovanju križanke v poglavju 4.5.

Meni pod križanko je sestavljen iz treh gumbov. Prvi je namenjen prikazu panelnega menija, drugi tipkovnice, tretji pa ima spremenljivo delovanje glede na izbrani način igranja. V enoigralskem načinu deluje kot gumb za izbris križanke iz podatkovne baze, v večigralskem načinu pa služi prikazu uporabnikovih točk.

```
<div id="GamePage" data-role="page" data-theme="a">
  <!-- panelni meni za enoigralski način -->
  <div id="GamePanelSingle" data-role="panel" data-theme="a">
    </div>
    <!-- panelni meni za večigralski način-->
  <div id="GamePanelMulti" data-role="panel" data-theme="a">
    </div>
  <!--  opis besede -->
  <div id="GameHeader" class="top-fixed orange-background">
    <div id="DescriptionWrapper">
      <div id="Description" class="white-box"></div>
    </div>
  </div>
  <!-- križanka -->
  <div id="CrosswordContainer">
    <span><input id="HiddenInput" data-role="none" /></span>
    <div id="Crossword"></div>
  </div>
  <!--  meni -->
  <div id="GameFooter" data-role="navbar">
```

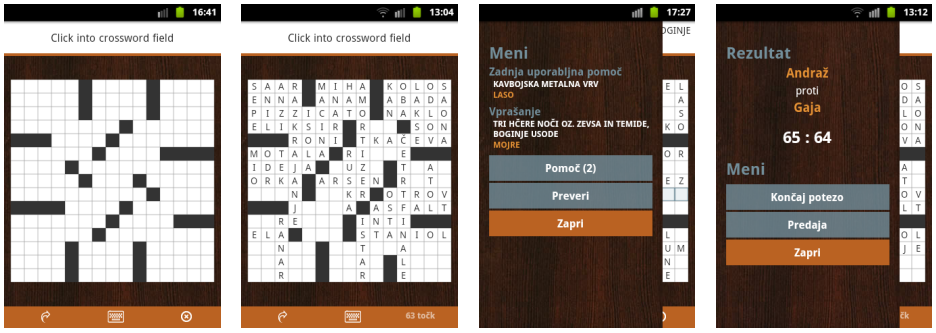
```
21      <ul>
22        <li><a id="GamePanel" href="#" data-iconshadow="false" data-icon="
              menu"></a></li>
23          <li><a id="Keyboard" href="#" data-iconshadow="false" data-icon="
                keyboard"></a></li>
24          <li><a id="PointsDelete" href="#" data-iconshadow="false" data-
                icon="delete"><p id="MenuPoints" class="pull-up-text"></p></a>
                </li>
25    </div>
26    <!-- pojavno okno -->
27    <div id="PopupGame" data-role="popup" data-overlay-theme="a" data-
          dismissible="false" data-add-back-btn="true">
28    </div>
29  </div>
```

Programska koda 4.13: Struktura HTML strani s križanko. Na začetku sta ustvarjena panelna menija, nato polje za prikaz vprašanja, križanka, meni in pojavno okno. Strukturi panelnih menijev in pojavnega okna nista prikazani v celoti.

Vsak igralni način ima svoj panelni meni. Oba sta prikazana na sliki 4.8. Uporabil sem dva, ker imata različni strukturi HTML, kar vpliva na prikaz informacij. Tako v enoigralskem kot večigralskem načinu se panelni meni odpre s pritiskom na skrajno levi gumb menija pod križanko. Oba panelna menija imata gumb za zaprtje.

Enoigralski panelni meni prikazuje vprašanje trenutno izbrane besede in besedo, pri kateri je bila nazadnje uporabljena pomoč. Vsebuje še gumba *Pomoč* in *Preveri*. Če uporabnik pritisne na gumb Pomoč, se mu v panelnem meniju, pod vprašanje trenutno izbrane besede, izpiše odgovor. Odgovor je napisan z oranžno, vprašanje pa z belo barvo. Ko pomoč uporabi, se na gumbu popravi napis, ki sporoča, kolikokrat se še lahko uporabi pomoč. Ob pritisku na gumb Preveri, se preveri pravilnost križanke in se prikaže pojavno okno z obvestilom.

Ko uporabnik pritisne gumb za pomoč, se najprej preveri, če je že preseženo število dovoljenih uporab pomoči. Če ni, se izpiše odgovor, ki ga mora uporabnik vnesti. Uporaba pomoči se zabeleži. Popravi se napis na gumbu, ki



Slika 4.8: Na prvi sliki so prikazani polje z vprašanjem, križanka in meni v enoigralskem načinu. Na drugi sliki je prikazan meni v večigralskem načinu. Na tretji sliki je prikazan panelni meni v enoigralskem, na četrti sliki pa v večigralskem načinu.

označuje, kolikokrat je bila pomoč uporabljena. Uporabljena pomoč se zabeleži v poseben objekt. V njem je zabeležena zgodovina uporabe pomoči za izbrano križanko. Ko uporabnik zapusti zaslon s križanko, se vsebina tega objekta pretvori v nizi in se shrani v podatkovno bazo.

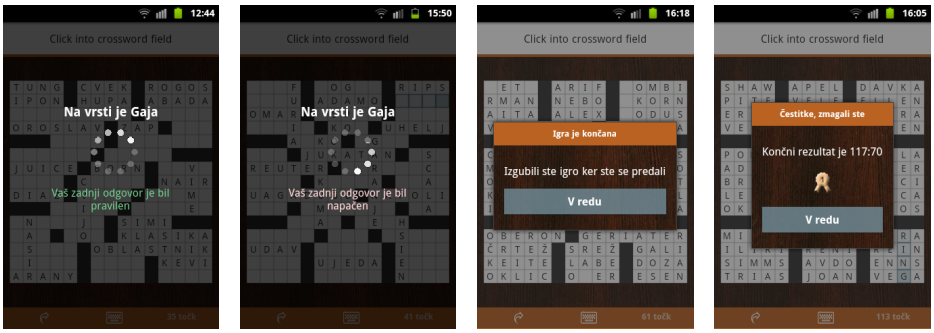
Pri preverjanju pravilnosti ni nobenih omejitev. Po uporabnikovem pritisku na gumb se preveri pravilnost vseh črk v križanki. To sem naredil tako, da se aplikacija sprehodil čez seznam vseh besed v križanki in za vsako črko besede preveri, če je enaka tisti, ki jo je vpisal uporabnik. Če je črka napačna se polje obarva rdeče. Poleg tega se beleži tudi število ujemanj. Če ni bila vpisana nobena črka, se to šteje kot napačen vnos (le da se polje ne obarva). Na koncu se preveri, če je število vseh pravilnih vnosov enako številu vseh črk. Če je enako, je uporabnik pravilno rešil križanko, sicer pa ne. V obeh primerih se prikaže pojavno okno z ustreznim obvestilom.

Večigralski panelni meni prikazuje trenutni rezultat igre in vsebuje še gumba *Končaj potezo* in *Predaja*. Ko uporabnik vnese svojo besedo pritisne gumb Končaj potezo. S pritiskom na gumb Predaja se uporabnik preda.

Ko uporabnik vpiše odgovor na vprašanje v križanko, mora na strežnik poslati podatke o svoji potezi. To stori s pritiskom na gumb *Končaj potezo*. Na strežnik se pošlje zahteva Ajax z njegovim odgovorom. Zaslon križanke se

takrat zaklene, da vnos v križanko ni več mogoč. Ta zaslon ga obvesti, če je bil njegov odgovor pravilen, in sporoča, da je na vrsti nasprotnik. V tem času se sproži časovnik, ki na deset sekund preverja, če je nasprotnik že končal svojo potezo. Časovnik na strežnik pošlje zahtevo Ajax za prenos zadnje poteze. Če je ta različna od poteze uporabnika, to pomeni, da je nasprotnik končal svojo potezo. V križanko se vnese nasprotnikov odgovor, popravi se točke in zaklenjeni zaslon se odstrani. Zaklenjeni zaslon in obvestili o koncu igre so prikazani na sliki 4.9.

Če uporabnik pritisne gumb *Predaja*, se igra konča, ker se je predal. Na strežnik se pošlje zahteva Ajax, ki sporoča predajo. Pojavi se pojavno okno z obvestilom.



Slika 4.9: Na prvi in drugi sliki je prikazan zaklenjeni zaslon z obvestilom o pravilnosti zadnje poteze uporabnika. Na tretji in četrti sliki sta prikazani pojavni okni z obvestiloma o koncu igre.

4.4.6 Nalagalni zaslon

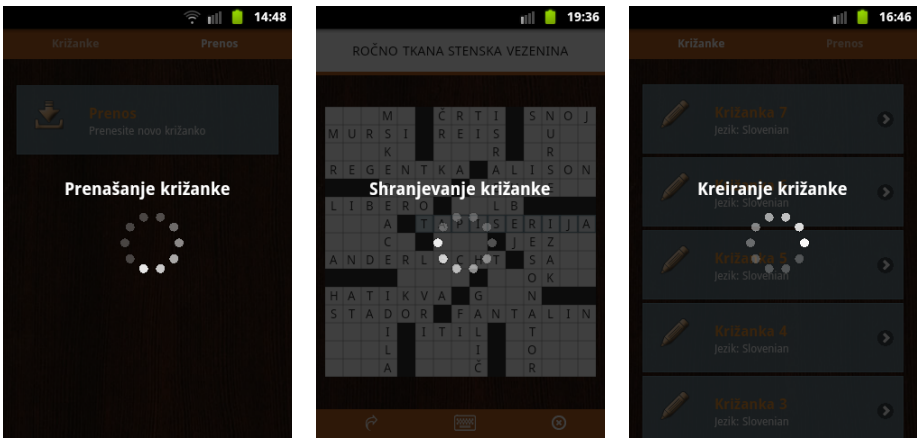
Ker se v aplikaciji prenašajo podatki na strežnik in s strežnika, v podatkovno bazo in iz nje, sem dodal nalagalni zaslon (angl. loading screen). Ta se prikaže medtem, ko se prenašajo ali shranjujejo podatki. Tako uporabnik ve, da mora malo počakati. V programski kodi 4.14 je prikazana struktura HTML nalagalnega zaslona.

```
1 <div id="ScreenLock" class="screen-lock">
2     <div class="screenlock-content">
3       <div class="screenlock-text">
```

```
4         <h3 id="ScreenLockTitle" class="ui-title"></h3>
5       </div>
6       <div class="spinner">
7         <div id="Spinner"></div>
8       </div>
9       <div class="screenlock-text">
10        <p id="LastWordText" class="ui-title"></p>
11      </div>
12    </div>
13  </div>
```

Programska koda 4.14: Struktura HTML nalagalnega zaslona. Zaslon se prikaže med prenosom in shranjevanjem podatkov ter kreiranjem križanke.

Zaslon sem zasnoval tako, da poleg animacije prikaže različna sporočila. Zaslon se prikaže z ukazom *CONFIG.ScreenLock.css('visibility','visible');*. Ta ukaz se uporabi tudi za skritje zaslona, le da se stilska lastnost vidljivost (angl. visibility) nastavi na skrito (angl. hidden). Slika 4.10 prikazuje nalagalni zaslon.



Slika 4.10: Slika prikazuje nalagalni zaslon med prenosom križanke s strežnika, shranjevanjem križanke v podatkovno bazo in med kreiranjem križanke.

4.5 Križanka

Križanke v aplikaciji so ustvarjene in shranjene na strežniku. V enoigralskem in večigralskem načinu se križanke prenesejo s strežnika. V enoigralskem načinu se prenesene križanke shranijo v podatkovno bazo. Vse križanke imajo enako strukturo. Med seboj se razlikujejo le v prisotnosti oziroma odsotnosti strukture *media*, ki vsebuje podatke o slikah v križanki.

4.5.1 Križanke v formatu JSON

Uporabljene križanke so ustvarjene z generatorjem križank, ki se nahaja na strežniku. Aplikacija dostopa do njega in prenaša ustvarjene križanke. Vse križanke imajo enako strukturo. Ta struktura je prikazana v programski kodi 4.15. Struktura se začne z nekaj splošnimi podatki o križanki. To so *id*, *velikost križanke* (število vrstic in stolpcev) in *uporabljeni slovar* oziroma jezik križanke. Sledi tabela *Lines*, ki predstavlja kvadratke križanke. Vrednosti v njej povedo, kakšno vlogo ima kvadratek v križanki. Vrednosti so lahko 0, 2 in 6. 0 pomeni črn kvadratek, 2 pomeni navaden bel oziroma vnosen kvadratek in 6 sliko. Sledi tabela *Words*. Tabela vsebuje seznam vseh besed v križanki. Vsaka beseda vsebuje podatek o svoji lokaciji v križanki, njeni usmerjenosti (vodoravno ali navpično), vrednosti (beseda) in pripadajoče vprašanje.

Če križanka vsebuje slike, sledi tabela *Media*. Vsaka slika ima podatek o svoji lokaciji v križanki, velikosti ter ime datoteke slike. Vse slike iste križanke se na strežniku nahajajo v skupnem imeniku.

```
1  {
2      "ID":0,
3      "Width":5,
4      "Height":5,
5      "Dictionary":{
6          "ID":0,
7          "Name":"Slovenian"
8      },
9      "Lines":[
10         "2,2,2,2,2",
```

```
        "2,2,2,2,2",
        "2,2,2,2,2",
        "2,2,2,2,2",
        "2,2,2,2,2"
    ],
    "Words":[
        {
            "ID":1,
            "Horizontal":true,
            "X":0,
            "Y":0,
            "Word":"VIDEO",
            "Description":"VIDEOREKORDER"
        },
        {
            "ID":2,
            "Horizontal":false,
            "X":0,
            "Y":0,
            "Word":"VEGRI",
            "Description":"SLOVENSKA PISATELJICA SAŠA"
        },
        {
            "ID":3,
            "Horizontal":false,
            "X":1,
            "Y":0,
            "Word":"IGRAM",
            "Description":"ŠČIT, NAPRAVA ZA LOV NA JEREBICE"
        },
        {
            "ID":4,
            "Horizontal":false,
            "X":2,
            "Y":0,
            "Word":"DIADA",
            "Description":"TENZOR DRUGE STOPNJE V MATEMATIKI"
        },
        {
            "ID":5,
            "Horizontal":false,
            "X":3,
            "Y":0,
            "Word":"ESSEN",
            "Description":"MESTO V NEMČIJI"
        },
        {
```

```
58              "ID":6,
59              "Horizontal":false,
60              "X":4,
61              "Y":0,
62              "Word":"OTAVA",
63              "Description":"REKA V SLOVAŠKI REPUBLIKI"
64          },
65          {
66              "ID":7,
67              "Horizontal":true,
68              "X":0,
69              "Y":1,
70              "Word":"EGIST",
71              "Description":"ŠPARTANSKI TIESTOV SIN"
72          },
73          {
74              "ID":8,
75              "Horizontal":true,
76              "X":0,
77              "Y":2,
78              "Word":"GRASA",
79              "Description":"VEJE ZA POKRIVANJE KOPE"
80          },
81          {
82              "ID":9,
83              "Horizontal":true,
84              "X":0,
85              "Y":3,
86              "Word":"RADEV",
87              "Description":"HRVAŠKA OPERNA PEVKA MARIJANA"
88          },
89          {
90              "ID":10,
91              "Horizontal":true,
92              "X":0,
93              "Y":4,
94              "Word":"IMANA",
95              "Description":"NAJVIŠJE BOŽANSTVO LJUDSTVA BANJARVANDA V URANDI"
96          }
97      ]
98  }
```

Programska koda 4.15: Primer osnovne križanke velikosti 5x5 v formatu JSON, brez slik.

4.5.2 Izris križanke

Izris križanke se izvede, ko uporabnik v enoigralskem načinu izbere križanko ali ko v večigralskem načinu izbere oziroma ustvari novo igro. V programski kodi 4.16 je prikazan izris križanke oziroma izgradnja njene strukture HTML in dodajanje stilskih predlog.

Križanka se zgradi tako, da se za vsako vrstico v tabeli *Lines* ustvari element HTML, ki predstavlja vrstico v križanki. Za vsako vrednost v tabeli pa se ustvari element HTML, ki predstavlja kvadratek v križanki. Vrstice z dodanimi kvadratki se dodajo elementu z id-jem Crossword. Tako se zgradi struktura križanke. Tabela Lines lahko vsebuje tri različne vrednosti, ki določajo vlogo kvadratka v križanki. Več o vrednostih v tabeli in njihovem pomenu je razloženo v poglavju 4.5.1.

```
1  draw: function(){
2    var line;
3    for (var i=0;i<APP.crossword.data.Height;i++){
4      $('<div id="line-'+i+'" class="line"></div>').appendTo(CONFIG.Crossword);
5      line = APP.crossword.data.Lines[i].split(',');
6      for (var j=0;j<APP.crossword.data.Width;j++){
7        if (line[j] === '0'){
8          $('<div id="'+j+'-'+i+'" class="field blank"><div></div></div>').
               appendTo('#line-'+i);
9          CROSSWORD_DATA.gridData[j][i]=null;
10       }
11       else if (line[j] === '2'){
12         $('<div id="'+j+'-'+i+'" class="field char"><div></div></div>').
               appendTo('#line-'+i);
13         CROSSWORD_DATA.gridData[j][i]={horizontalWord:'',verticalWord:''};
14       }
15       else if (line[j] === '6'){
16         $('<div id="'+j+'-'+i+'" class="field picture"><div></div></div>').
               appendTo('#line-'+i);
17         CROSSWORD_DATA.gridData[j][i]=null;
18       }
19     }
20   }
21 }
```

Programska koda 4.16: Ustvarjanje križanke. Najprej se ustvarijo vrstice, katerim se dodajo kvadratki. Kvadratkom se dodajo še razredi za stilske predloge.

Vsaki ustvarjeni vrstici se dodajo kvadratki. Vsak kvadratek ima določen id, ki je enak njegovi lokaciji v križanki (npr. element v 11. stolpcu in 5. vrstici ima id 11-5). Glede na vrednosti v tabeli Lines se kvadratkom določi stilske predloge. To sem naredil s prireditvijo ustreznih razredov kvadratkom. Uporabljene stilske predloge za križanko so prikazane v programski kodi 4.17.

```
.line {
  width: 100%;
    position: relative;
    clear: both;
}
.field {
  background: white;
  float: left;
  text-transform: uppercase;
  text-align: center;
}
.char {
    -webkit-box-shadow: inset 0px 0px 1px 0px #333;
    box-shadow: inset 0px 0px 1px 0px #333;
}
.selected {
    -webkit-box-shadow: inset 0px 0px 2px 3px #aecfdd;
    box-shadow: inset 0px 0px 2px 3px #aecfdd;
    background: #ecf2f5;
}
.marked {
    -webkit-box-shadow: inset 0px 0px 2px 3px #8b80d7;
    box-shadow: inset 0px 0px 2px 3px #8b80d7;
    background: #ecf2f5;
}
.blank {
    background: #333;
}
.picture {
    background: #0094ff;
}
.image {
    z-index: 1;
    position: absolute;
}
```

Programska koda 4.17: Stilske predloge za izris križanke.

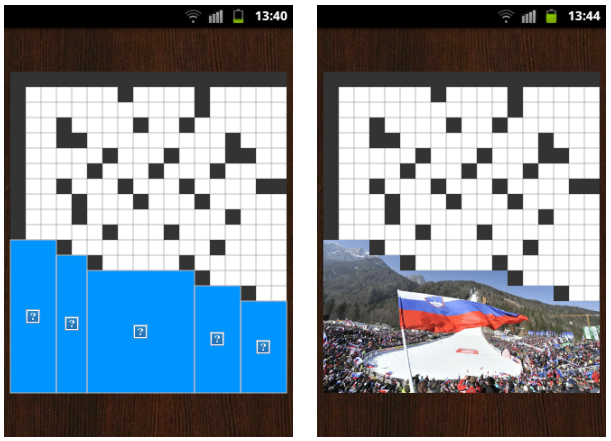
V aplikaciji sem želel prikazati celotno križanko na zaslon. Zato je bilo potrebno programsko določiti velikost kvadratkov in križanke. Glede na višino in širino zaslona mobilne naprave ter glede na število vrstic in stolpcev križanke se izbere primerno velikost kvadratka križanke. Velikost je izbrana tako, da zapolni čim večji del zaslona, pri čemer mora ob robovih ostati nekaj prostora, da je prisoten občutek, kot da je križanka položena na mizo. Prav tako je bilo potrebno določiti velikost vsebovalnika križanke. Ta se razprostira med poljem za prikaz vprašanja in menijem. Vsebovalnik nima določenega nobenega ozadja, zato ni viden. V programski kodi 4.18 je prikazano nastavljanje nekaterih lastnosti križanke.

```
1  setParameters: function(){
2    var pageWidth=$(window).width(),
3      CrosswordMaxHeight=CONFIG.AppHeight-headerHeight-CONFIG.Footer.height(),
4      headerHeight=CONFIG.Header.height(),
5      CrosswordMaxWidth=pageWidth;
6    var fieldSize1=Math.floor(CrosswordMaxHeight/APP.crossword.data.Height),
7      fieldSize2=Math.floor(CrosswordMaxWidth/APP.crossword.data.Width);
8    CONFIG.FieldSize=(fieldSize1<fieldSize2) ? fieldSize1 : fieldSize2;
9    CONFIG.FieldSize=Math.round(CONFIG.FieldSize*0.95);
10   CONFIG.Field = $('.field');
11   CONFIG.Field.css('width',CONFIG.FieldSize)*0.8;
12   CONFIG.Field.css('height',CONFIG.FieldSize)*0.8;
13   CONFIG.ContainerHeight=CONFIG.AppHeight-headerHeight-CONFIG.Footer.height
         ();
14   CONFIG.ContainerWidth=pageWidth;
15   CONFIG.Container.css('height',CONFIG.ContainerHeight);
16   CONFIG.Container.css('width',CONFIG.ContainerWidth);
17   CONFIG.CrosswordWidth=CONFIG.FieldSize*APP.crossword.data.Width;
18   CONFIG.CrosswordHeight=CONFIG.FieldSize*APP.crossword.data.Height;
19   CONFIG.Crossword.css('width',CONFIG.CrosswordWidth);
20   CONFIG.Crossword.css('height',CONFIG.CrosswordHeight);
21   CONFIG.Container.css('margin-top',headerHeight);
22   var posCross=(CONFIG.Container.height()-CONFIG.Crossword.height())/2-
         CONFIG.FieldSize;
23   CONFIG.Crossword.css('margin-top',(CONFIG.Container.height()-CONFIG.
         Crossword.height()))/2-CONFIG.FieldSize);
24   var textSize=Math.floor(CONFIG.FieldSize*0.7),
25       margin=Math.floor((CONFIG.FieldSize-textSize)/2);
26   $('.field').children().css('font-size',textSize);
27   $('.field').children().css('margin-top',margin);
28   var keyboard=window.localStorage.getItem("keyboard");
29   KEYBOARD.height=parseInt(keyboard);
```

```
30      CONFIG.ContainerZoomWithKeyboardHeight=CONFIG.AppHeight-KEYBOARD.height-
            headerHeight;
31  }
```

Programska koda 4.18: Nastavitev lastnosti križanke. Nastavi se velikost kvadratka križanke, velikost križanke, velikost vsebovalnika križanke, velikost pisave v kvadratkih in odmiki elementov.

Križanke lahko vsebujejo tudi slike. Če jih, se v strukturi JSON nahaja tudi tabela *Media*. V njej so podatki o lokaciji slik, njihovi velikosti in imenih. V enoigralskem načinu se, poleg križanke, prenesejo tudi slike. Te se shranijo na spominsko kartico naprave. Če križanka vsebuje slike, se za vsako sliko v tabeli Media ustvari značka *img*. Ta značka označuje slike. Znački se doda id, stilske predloge, pot do slike na spominski kartici in nastavi se velikost slike. Primer križanke s sliko je prikazan na sliki 4.11.



Slika 4.11: Prva slika prikazuje križanko pred dodajanjem slik. Modro obarvani kvadratki označujejo, kje bodo slike. Vidi se, da je končna slika sestavljena iz več manjših slik. Vsaka slika predstavlja en vnos v tabeli Media. Druga slika prikazuje križanko s končno sliko.

4.5.3 Povečava križanke

Nekatere mobilne naprave imajo manjše zaslone ali pa imajo majhno ločljivost. Na njih so križanke majhne in črke v kvadratkih so slabo vidne. Zato sem dodal možnost povečave križanke.

Križanka se poveča ali zmanjša z dvojnim pritiskom (angl. double tap) nanjo. Ko pride do dvojnega pritiska, se preveri, če je potrebno križanko povečati ali zmanjšati. V primeru povečanja se križanka poveča za enkrat. Če želimo, da se križanka zmanjša, se zmanjša na privzeto velikost. Na sliki 4.12 je prikazana razlika med normalno in povečano križanko.

Slika 4.12: Prva slika prikazuje križanko v normalni velikosti, druga pa v povečani velikosti.

Križanka se poveča z ukazom *CONFIG.Crossword.css('zoom','2')*. Ob povečavi se spremenita odmik križanke od zgornjega roba in velikost skritega vnosnega polja. Če je bila beseda v križanki izbrana pred povečavo, se križanka premakne tako, da je izbrana beseda vidna. Telesu dokumenta se priredi stilska predloga, ki prepreči nenadne premike križanke gor ali dol. To sem naredil z ukazom *$(document.body).css('overflow','hidden')*.

Ob zmanjšanju križanke se zmanjša skrito vnosno polje, spremenita se odmika vsebovalnika in križanke ter spremeni se velikost vsebovalnika. Spremeni se tudi stilska predloga telesa dokumenta, ki zopet omogoči premike.

V programski kodi 4.19 je prikazano povečanje in zmanjšanje križanke.

```
1  zoomIn: function(){
2    CONFIG.Crossword.css('zoom','2');
3    ZOOM.zoom = true;
4    $(document.body).css('overflow','hidden');
5    var pullUp=(CONFIG.Crossword.position().top+(CONFIG.Crossword.offset().top
         -CONFIG.Container.offset().top));
6    CONFIG.Crossword.css('margin-top',-pullUp);
7    CONFIG.HiddenInput.css('width',CONFIG.FieldSize*2);
8    CONFIG.HiddenInput.css('height',CONFIG.FieldSize*2);
9    ZOOM.move = true;
10   if (CROSSWORD.SelectedWord !== null)
11   {
12     KEYBOARD.moveCrossword();
13   }
14   ZOOM.move = false;
15 }
16 zoomOut: function(){
17   ZOOM.zoom = false;
18   $(document.body).css('overflow','scroll');
19   CONFIG.Crossword.css('zoom','1');
20   CONFIG.HiddenInput.css('width',CONFIG.FieldSize);
21   CONFIG.HiddenInput.css('height',CONFIG.FieldSize);
22   CONFIG.Container.css('height',CONFIG.ContainerHeight);
23   CONFIG.Container.css('margin-top',CONFIG.Header.height());
24   CONFIG.Crossword.css('margin-top',(CONFIG.Container.height()-CONFIG.
         Crossword.height())/2-CONFIG.FieldSize);
25   CONFIG.HiddenInput.css('left',0);
26   CONFIG.HiddenInput.css('top',0);
27 }
```

Programska koda 4.19: V primeru povečave križanke se izvede funkcija *zoomIn*, v primeru zmanjšanja pa funkcija *zoomOut*.

4.5.4 Premikanje križanke

Povečana križanka je tako velika, da jo je potrebno premikati, če jo želimo videti v celoti. Premikanje povečane križanke sem omogočil z dodano stilsko predlogo *overflow: hidden* vsebovalniku križanke. Poleg tega sem vsebovalnik povezal z dogodkoma *touchstart* in *touchmove*. Dogodek touchstart se sproži ob dotiku. Takrat se shranita koordinati dotika in koordinati križanke. Ob dogodku touchmove se spet zabeleži koordinati dotika. Iz starih in novih ko-

ordinat dotika se izračuna, za koliko in kam je potrebno premakniti križanko. Premik se izvede tako, da se premakne vsebina vsebovalnika. V tem primeru križanka zaseda celoten vsebovalnik. Premik vsebovalnikove vsebine v smeri osi x se izvrši z ukazom *scrollTop(y)*, v smeri osi y pa z ukazom *scrollLeft(x)*. Parametra x in y v funkcijah scrollTop in scrollLeft pomenita lokacijo, na katero naj se vsebina premakne. V programski kodi 4.20 sta prikazana dogodka za premik križanke, na sliki 4.13 pa je prikazan primer premika.

Slika 4.13: Sliki prikazujeta premik križanke, ko je ta povečana.

```
CONFIG.Container.bind('touchstart',function(event){
  if (ZOOM.zoom){
    var e=event.originalEvent;
    CROSSWORD.TouchX=e.targetTouches[0].pageX;
    CROSSWORD.TouchY=e.targetTouches[0].pageY;
    CROSSWORD.TouchStartX=CONFIG.Container.scrollLeft();
    CROSSWORD.TouchStartY=CONFIG.Container.scrollTop();
  }
});

CONFIG.Container.unbind().bind('touchmove',function(event){
  if (ZOOM.zoom){
    var e=event.originalEvent;
    e.preventDefault();
    var x=CROSSWORD.TouchStartX-(e.targetTouches[0].pageX-CROSSWORD.TouchX),
        y=CROSSWORD.TouchStartY-(e.targetTouches[0].pageY-CROSSWORD.TouchY);
    if ((Math.abs(e.targetTouches[0].pageY-CROSSWORD.TouchY)<3*CONFIG.
        FieldSize)&&(Math.abs(e.targetTouches[0].pageX-CROSSWORD.TouchX)>3*
        CONFIG.FieldSize)){
```

```
18          CONFIG.Container.scrollLeft(x);
19          KEYBOARD.hiddenInputPosX=x;
20          CONFIG.HiddenInput.css('left',KEYBOARD.hiddenInputPosX);
21        }
22        else if ((Math.abs(e.targetTouches[0].pageY-CROSSWORD.TouchY)>3*CONFIG.
              FieldSize)&&(Math.abs(e.targetTouches[0].pageX-CROSSWORD.TouchX)<3*
              CONFIG.FieldSize)){
23          CONFIG.Container.scrollTop(y);
24          KEYBOARD.hiddenInputPosY=y;
25          CONFIG.HiddenInput.css('top',KEYBOARD.hiddenInputPosY);
26        }
27        else{
28          CONFIG.Container.scrollTop(y);
29          CONFIG.Container.scrollLeft(x);
30          KEYBOARD.hiddenInputPosX=x;
31          KEYBOARD.hiddenInputPosY=y;
32          CONFIG.HiddenInput.css('left',KEYBOARD.hiddenInputPosX);
33          CONFIG.HiddenInput.css('top',KEYBOARD.hiddenInputPosY);
34        }
35      }
36  });
```

Programska koda 4.20: Premik križanke. V dogodkih *touchstart* in *touchmove* se zabeležita koordinati dotika, iz katerih se izračuna, za koliko in kam je potrebno premakniti križanko. Za pohitritev premika prižanke je potrebno v dogodku *touchmove* preprečiti privzeto akcijo na dogodek.

4.5.5 Izbira besede

Besedo se izbere tako, da se pritisne na vnosni kvadratek križanke. Če izbrani kvadratek pripada dvema besedama, se preveri, če se katera od njiju začne v tem kvadratu. Če se, se ta beseda izbere. Če se obe besedi začneta v tem kvadratku, ima prednost vodoravna beseda. Če se nobena od besed ne začne v kvadratku, se izbere besedo glede na vrednost spremenljivke ki določa smer (vodoravno ali navpično). Vrednost te spremenljivke se spremeni, če se isto polje izbere dvakrat zaporedoma.

Za hitrejši odziv na izbiro besede sem ustvaril dve tabeli. Obe se ustvarita,

ko se ustvari križanka. Prva je enodimenzionalna tabela *wordList*, ki vsebuje vse besede križanke in podatke o njih. Podatki, ki so shranjeni v tabeli, so id besede, beseda, vprašanje in polja besede (id-ji kvadratkov, ki so del besede). Vsaka beseda v tej tabeli ima svoj indeks. Te indekse sem shranil v dvodimenzionalno tabelo *gridData*, ki je enakih dimenzij kot križanka. Vsak element tabele ima dve vrednosti. Prva je indeks vodoravne besede in druga indeks navpične besede, ki se nahaja na tem mestu v križanki.

Ko uporabnik pritisne na polje, se iz njegovega id-ja pridobi indeksa, ki se ju uporabi za dostop do tabele gridData. Primer id 5-11 pomeni peti stolpec in enajsta vrstica v tabeli. Iz tabele se preneseta indeksa za vodoravno in navpično besedo v tabeli wordList. Iz te tabele se nato preneseta obe besedi. Nato se določi, katera beseda bo izbrana po postopku, ki je opisan na začetku podpoglavja 4.5.5.

Celotno besedo se označi z eno barvo, izbran kvadratek, na katerega je uporabnik pritisnil, pa z drugo. V programski kodi 4.21 je prikazano označevanje izbrane besede. V polje nad križanko se izpiše vprašanje za izbrano besedo. Ko uporabnik izbere novo besedo, se stara beseda odznači in označi se nova.

```
1  markField: function(){
2    if (CROSSWORD.SelectedField !== ''){
3      $('#'+CROSSWORD.SelectedField).addClass('selected');
4    }
5  }
6
7  markWord: function(){
8    for (var i=0;i<CROSSWORD.SelectedWord.fields.length;i++){
9      CROSSWORD.SelectedWord.fields[i].addClass('marked');
10   }
11 }
```

Programska koda 4.21: Označitev izbranega kvadratka in besede. Označitev se naredi tako, da se vsakemu kvadratku besede doda ustrezno stilsko predlogo. Besedo se odznači tako, da se kvadratkom odvzame stilsko predlogo.

4.5.6 Vnos črk

Ko je beseda izbrana in je prikazana tipkovnica, je omogočen vnos črk. S pritiskom na tipko tipkovnice se vnese črka v izbrani kvadratek. Nato se izbrano polje premakne za eno mesto naprej. Pri brisanju se, če izbrani kvadratek ni prazen, izbriše črka, izbrano polje pa ostane enako. Če je polje prazno, se izbrani kvadratek premakne za eno mesto nazaj. Tako sem s tipko za brisanje realiziral brisanje in pomikanje nazaj. Slika 4.14 prikazuje vnos črk v križanko.

Slika 4.14: Prva in druga slika prikazujeta vnos črk pri normalni velikosti križanke, tretja in četrta slika pa prikazujeta vnos pri povečani križanki.

Da je brisanje delovalo, kot je opisano v zgornjem odstavku, sem moral uporabiti trik. Ko je postalo vnosno polje aktivno, sem vanj vnesel trideset presledkov, nato pa se je prikazala tipkovnica. Teh trideset presledkov sem uporabil zato, da lahko uporabnik uporablja gumb za brisanje in premikanje. Vnos sem namreč naredil tako, da če je dolžina vnosa v vnosnem polju večja od dolžine, ki je bila pred pritiskom na gumb tipkovnice, se zadnja vnesena črka doda, drugače pa se izbriše vsebina izbranega kvadratka. Če vnosno polje ne bi že v začetku vsebovalo nekaj znakov, brisanje in premikanje nazaj ne bi bilo mogoče.

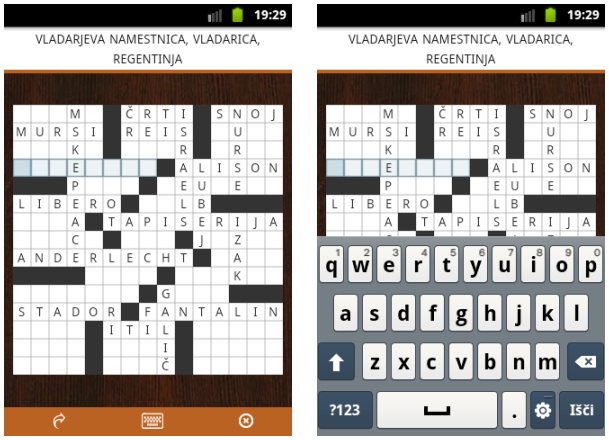
Razlika med vnosom črk v enoigralskem in večigralskem načinu je v zaklepu besede. V večigralskem načinu se, ko uporabnik vpiše vsaj eno črko v besedo, ta zaklene. Zaklene se tako, da ni več mogoče izbrati drugih besed.

To sem naredil, ker lahko uporabnik v eni potezi vpiše le eno besedo. Če izbriše svoje vnose v besedi, se ponovno omogoči izbiranje drugih besed.

4.6 Tipkovnica

Na mobilnih napravah se tipkovnica prikaže, ko uporabnik pritisne oziroma izbere vnosno polje. V križanki nisem želel imeti vnosnega polja, zato sem ga skril pod križanko. To sem naredil z določitvijo lastnosti *z-index* v stilski predlogi. Ta določa, kateri elementi bodo postavljeni v ospredje. Tako sem omogočil prikazovanje tipkovnice, ko je postalo vnosno polje aktivno. Vnosno polje se aktivira programsko in ne s pritiskom vanj.

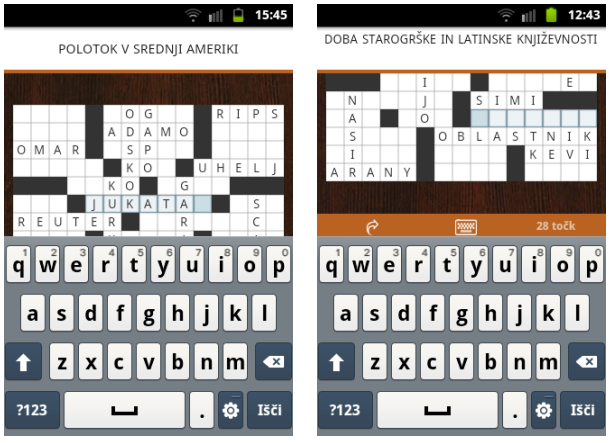
Tipkovnica se prikaže, ko uporabnik pritisne na gumb za njen prikaz. Takrat postane skrito vnosno polje aktivno (v nadaljevanju le vnosno polje). Vnosno polje se aktivira programsko z uporabo funkcije *focus()*. Slika 4.15 prikazuje križanko z in brez tipkovnice. Ko začne uporabnik vnašati črke, se te vnesejo v vnosno polje. Vnos črk se zazna z dogodkom *input*, ki ga sproži sprememba vsebine vnosnega polja. Ali gre za brisanje ali za vnos črke, se zazna po dolžini vnosa v vnosnem polju. Če je dolžina večja, kot je bila ob prejšnji sprožitvi istega dogodka, se zadnja črka doda v trenutni kvadratek, če pa je dolžina manjša, se izbriše vsebina trenutnega kvadratka.



Slika 4.15: Prva slika prikazuje križanko brez, druga pa z tipkovnico.

Tipkovnica lahko prekriva izbrano besedo, zato je potreben premik križanke. Način premika križanke je odvisen od njene velikosti. Če križanka ni povečana, se naredi premik telesa (element HTML). Če je križanka povečana, se naredi premik vsebine vsebovalnika križanke.

Ko križanka ni povečana, se preveri, ali se izbrana beseda nahaja pod tipkovnico. To se preveri tako, da se primerja pozicijo izbrane besede in pozicijo tipkovnice. Pozicija tipkovnice se izračuna s pomočjo lokalno shranjenega podatka o njeni višini. Če se beseda nahaja pod tipkovnico, se celotno telo aplikacije pomakne navzgor. Pomik se naredi z uporabo ukaza *$('body').animate(scrollTop:y,delay)*. Ukaz premakne celotno telo na pozicijo, ki jo določa parameter y. Premik se izvrši samo v smeri osi y. Če je beseda vodoravna, bo vidna v celoti, če pa je navpična in dolga, potem ne bo vidna v celoti. V tem primeru bo viden del besede, ki pa bo vseboval trenutno izbran kvadratek. Ko se vnašajo črke in se izbrani kvadratek nahaja blizu zgornjega ali spodnjega roba, se spet naredi premik. Tako je vedno viden del besede, ki je v danem trenutku pomemben za uporabnika. V tem načinu se vnosno polje ne premika s križanko. Slika 4.16 prikazuje premik križanke, ko je ta pomanjšana.



Slika 4.16: Prva slika prikazuje besedo nad tipkovnico, zaradi katere premik križanke ni potreben. Druga slika prikazuje besedo pod tipkovnico, zaradi katere se je premaknila križanka navzgor.

Ko je križanka povečana in je tipkovnica prikazana, se višina vsebovalnika zmanjša na velikost vmesnega prostora med poljem za prikaz vprašanja in vrhom tipkovnice. Če je beseda izven vidnega polja, se vsebovalnik križanke premakne tako, da je viden čim večji del besede. Če je mogoče, se prikaže celotno besedo, drugače pa se prikaže tisti del besede, ki vsebuje izbrani kvadratek. Premiki vsebovalnika so narejeni z ukazoma *scrollTop(y)* in *scrollLeft(x)*. Premiki se izvršijo v smeri osi x in y. V primeru, ko je križanka povečana, se skrito vnosno polje premika na mesto izbranega kvadratka. S tem sem preprečil nenadne premike križanke proti skritemu vnosnem polju, ki se je nahajalo drugje kot aktiven kvadratek. Ko se tipkovnica skrije, se velikost vsebovalnika križanke spremeni nazaj na prvotno velikost. Slika 4.17 prikazuje premik križanke, ko je ta povečana.

Slika 4.17: Prva slika prikazuje besedo pod tipkovnico, zaradi katere bo potreben premik. Druga slika prikazuje, kako se je križanka premaknila, ker se je beseda nahajala pod tipkovnico.

Poglavje 5

Sklepne ugotovitve

V diplomski nalogi sem ustvaril aplikacijo, ki omogoča reševanje križank na mobilnih napravah. Aplikacija omogoča dva načina igranja. V enoigralskem načinu se rešuje križanke na klasičen način, v večigralskem načinu pa med sabo tekmujeta dva igralca, ki izmenično vnašata besede v križanko.

Za izdelavno aplikacije sem uporabil ogrodje PhoneGap. Prednost uporabe tega ogrodja je razvoj aplikacije za več mobilnih platform hkrati. Žal to velja le do določene mere. Za vsako mobilno platformo ostaja osnova, ki je ustvarjena s tehnologijami HTML, CSS in JavaScript, enaka, domorodni deli aplikacij pa se razlikujejo. Zato je potrebno za vsako platformo ustvariti svoj domorodni del aplikacije. To je še vedno boljše kot razvoj celotne aplikacije za vsako platformo posebej.

Pri razvoju sem naletel na kar nekaj težav. Prva težava je bila, kako prikazati tipkovnico. Tipkovnica se na mobilnih napravah prikaže le, ko je vnosno polje aktivno. Vnosnega polja nisem mogel kar tako uporabiti, saj bi pokvaril videz križanke. Težavo sem odpravil tako, da sem vnosno polje skril pod križanko.

Druga težava je bila, kdaj prikazati tipkovnico. Sprva sem naredil aplikacijo tako, da se je ob uporabnikovem pritisku na kvadratek križanke prikazala tipkovnica. Ta je ob pritisku na drug kvadratek izginila in se nato spet pojavila. Ta prehod je bil zelo moteč. Do njega je prišlo, ker je za kratek čas

vnosno polje postalo neaktivno. To sem rešil tako, da sem dodal tipko za prikaz tipkovnice. Če se pritisne na drug kvadratek in je tipkovnica prikazana, se tipkovnica skrije in se ne prikaže, dokler uporabnik ne pritisne tipke za njen prikaz.

Težavo je predstavljalo tudi premikanje križanke, predvsem takrat, ko je križanka povečana in je prikazana tipkovnica. Ko je želel uporabnik videti spodnji del križanke, tega ni mogel, ker se križanka ni premaknila višje. Težavo sem rešil tako, da sem križanki nastavil večji odmik od spodnjega roba. Pri tej rešitvi se je pri premiku križanke videl del ozadja, ker se je križanka vedno preveč premaknila. Kakorkoli sem popravljal odmik je bil del ozadja viden. Na koncu sem obe težavi odpravil tako, da se je vsebovalnik križanke, ko se je prikazala tipkovnica, zmanjšal na velikost, ki je bila enaka prostoru med poljem za prikaz vprašanja in vrhom križanke. Tako je bilo mogoče premakniti križanko čisto do vrha brez vidnega ozadja.

Ob vnosu črk v križanko, ko je bila ta povečana, je prihajalo do nenadnih premikov križanke. Ugotovil sem, da se je premikalo celotno telo (značka telo v strukturi HTML) aplikacije. Zakaj je do tega prihajalo, nisem ugotovil, sem pa problem rešil z dodano stilsko predlogo, ki zaklene telo in s tem onemogoči premike. Ko križanka ni povečana, je ta predloga odstranjena.

Težavo je predstavljala tudi kombinacija premikanja križanke in vnosa črk, ko križanka ni povečana. Tudi v tem primeru prihaja do nenadzorovanega premikanja križanke. Problem sem poskušal rešiti s premikanjem vnosnega polja na mesto izbranega kvadratka. Na eni napravi je to rešilo problem, na drugih dveh pa ne. Poskusil sem tudi z dodajanjem enake stilske predloge kot pri povečani križanki. Ta rešitev ni delovala na nobeni napravi. Poskusil sem še nekaj možnih rešitev, a brez uspeha. Na koncu sem se odločil za rešitev, kjer je vnosno polje fiksno postavljeno v zgornji levi kot zaslona. Ta rešitev na eni napravi deluje, na drugi skoraj v celoti (nadležno premikanje se še pojavlja, vendar ne tako pogosto), na tretji pa ne.

Imel sem še nekaj manjših težav, kot na primer kako zagotoviti izvajanje programske kode šele potem, ko se izvedejo vse zahteve Ajax, kako prikazati

različno dolga vprašanja, kako dobiti višino tipkovnice in druge. Te težave so bile lažje rešljive od prej navedenih.

Aplikacija je dokončana, vendar pripravljena le za mobilno platformo Android. Razlog za to je pomanjkanje naprav, na katerih bi lahko testiral aplikacijo za druge platforme. Aplikacija deluje v celoti le na eni testni napravi, in sicer na mobilnem telefonu z Androidom verzije 2.3.3. Na podlagi tega sklepam, da aplikacija deluje na vseh novejših različicah platforme, ne morem pa tega trditi. Pojavijo se lahko specifične težave zaradi različnih različic spletnih brskalnikov, kot so se že med razvojem. To je še druga slabost razvoja aplikacij z ogrodjem PhoneGap. Razlike med brskalniki namreč privedejo razvijalce do sklepanja kompromisov, včasih celo do opustitve nekaterih lastnosti aplikacije.

V aplikaciji je še kar nekaj prostora za izboljšave. Prva bi bila vsekakor zagotovitev pravilnega delovanja vnosa črk in premikanja križanke brez nadležnih premikov na vseh napravah. Potrebna bi bila tudi omogočitev delovanja aplikacije v ozadju skupaj z uporabo sistemskih obvestil za obveščanje o dogodkih v igri (npr. nasprotnik je končal svojo potezo). Prav tako bi bilo potrebno dodelati izgled aplikacije, ki trenutno uporablja popravljene stilske predloge jQuery Mobile. Izgled je trenutno precej preprost, danes pa so uporabniki precej zahtevni in želijo uporabljati le očem prijetne aplikacije. Morda bi bila potrebna tudi izboljšava povečave križanke. Trenutno se križanka poveča ob dvojnem pritisku nanjo. Danes veliko aplikacij uporablja zvezno povečavo z uporabo gest.

Med izdelavo diplomske naloge sem se naučil mnogo novih stvari. Dobro sem se spoznal z JavaScript-om in knjižnicama jQuery in jQuery Mobile. Najbolj pomembno od vsega pa je spoznanje, da je mogoče razviti mobilne aplikacije tudi na drugačen način kot le z domorodnimi programskimi jeziki mobilnih platform.

Literatura

[1] Raymond Camden and Andy Matthews. *jQuery Mobile Web Development Essentials*. Packt Publishing, 2012.

[2] (2013) Tržni delež mobilnih platform. Dostopno na: `http://www.gartner.com/newsroom/id/2573415`.

[3] (2013) Podatkovni format JSON. Dostopno na: `http://www.json.org/`.

[4] Rohit Ghatol and Yogesh Patel. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Apress, 2012.

[5] Craig Grannell, Victor Sumner, and Dionysios Synodinos. *The Essential Guide to HTML5 and CSS3 Web Design (Essential Guides)*. friendsofED, 2012.

[6] Wesley Hales. *HTML5 and JavaScript Web Apps*. O'Reilly Media, 2012.

[7] (2013) Uporabljene ikone. Dostopno na: `http://www.brankic1979.com/icons`.

[8] (2013) Projekt jQuery Finger. Dostopno na: `https://github.com/ngryman/jquery.finger`.

[9] (2013) JavaScript knjižnica jQuery. Dostopno na: `http://jquery.com`.

[10] (2013) JavaScript knjižnica jQuery Mobile. Dostopno na: http://jquerymobile.com.

[11] (2013) Zgodovina križank. Dostopno na: http://en.wikipedia.org/wiki/Crossword.

[12] (2013) PhoneGap. Dostopno na: http://phonegap.com.

[13] (2013) Delež različic platforme Android. Dostopno na: http://developer.android.com/about/dashboards/index.html.

[14] (2013) Projekt spin.js. Dostopno na: http://fgnass.github.io/spin.js.

[15] John M Wargo. *PhoneGap Essentials: Building Cross-Platform Mobile Apps*. Addison-Wesley Professional, 2012.

[16] Nicholas C Zakas. *Professional javascript for web developers*. Wrox, 2011.