Web aplikacija logičke zagonetke Hashi.

Vrbić, Antonio

Master's thesis / Diplomski rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:951165

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

WEB APLIKACIJA LOGIČKE ZAGONETKE „HASHIWOKAKERO“

Diplomski rad

Antonio Vrbić

Osijek, 2019.

Sadržaj

1. UVOD 1
   1.1. Zadatak diplomskog rada 1
2. ZAGONETKA HASHIWOKAKERO (MOSTOVI) 2
   2.1. Pravila zagonetke 3
   2.2. Metode rješavanja 3
3. PROGRAMSKO RJEŠENJE WEB APLIKACIJE 4
   3.1. Korištene tehnologije i alati 4
      3.1.1. Programski jezici 4
      3.1.2. Programski okviri i biblioteke 6
   3.2. Implementacija algoritma za rješavanje zagonetke 8
      3.2.1. Strukture podataka 8
      3.2.2. Inicijalizacija rješavača 11
      3.2.3. Tijek rješavanja zagonetke 12
   3.3. Detekcija zagonetke iz slike 15
      3.3.1. Detekcija pozicija otoka 16
      3.3.2. Prepoznavanje broja na otoku 21
   3.4. Implementacija korisničkog sučelja 22
      3.4.1. Struktura aplikacije 22
      3.4.2. Komponente 25
4. TESTIRANJE I UPOTREBA WEB APLIKACIJE 31
   4.1. Testiranje algoritma rješavanja zagonetke 31
   4.2. Testiranje prepoznavanja zagonetke preko slike 32
   4.3. Testiranje korisničkog sučelja 34
      4.3.1. Brzina učitavanja web aplikacije 34
      4.3.2. Pregled datoteka web aplikacije 37

4.3.3. Prilagodljivost različitim veličinama zaslona 38
4.4. Upute za korištenje 40
4.4.1. Ručni unos podataka 40
4.4.2. Automatski unos podataka iz slike 42
5. ZAKLJUČAK 43
LITERATURA 44
POPIS TABLICA I SLIKA 46
SAŽETAK 48
ŽIVOTOPIS 49
PRILOZI (na CD-u) 50

1. UVOD

Glavni cilj ovog rada je izrada web aplikacije koja će uspješno riješiti *Hashiwokakero* zagonetku uz prethodno prepoznavanje uz pomoć kamere. Izrada web aplikacije podrazumijeva aplikaciju koja je prvenstveno prilagođena mobilnim uređajima s jednostavnim korisničkim sučeljem. Implementacijom web aplikacije mora biti ostvarena brza aplikacija koja ima svojstva izvorne aplikacije (engl. *native application*) te pokazati mogućnost korištenja naprednih biblioteka na web platformi koje prije nekoliko godina uopće nisu postojale.

Drugo poglavlje ovog rada bavi se teorijskim dijelom Hashiwokakero zagonetke što podrazumijeva metode rješavanja zagonetke. U trećem poglavlju opisano je potpuno programsko rješenje aplikacije u kojem su detaljno opisani korišteni programski jezici, okviri i biblioteke, gdje je najveća pozornost obraćena na programski okvir Angular koji pogoni web aplikaciju. Osim njega, opisane su *OpenCV* i *Tesseract* biblioteke koje omogućuju obradu slike i prepoznavanje znakova iz slike. Prikazana je implementacija algoritma za rješavanje zagonetke, korištene strukture podataka, inicijalizacija i tijek rješavanja zagonetke. Također, opisan je postupak prepoznavanja zagonetke iz predane slike korištenjem metoda obrade slike i prepoznavanja znakova. Četvrto poglavlje bazira se na testiranju izrađene web aplikacije. Prikazani su rezultati testiranja brzine rješavanja zagonetke, brzine prepoznavanja zagonetke te brzine učitavanje web aplikacije.

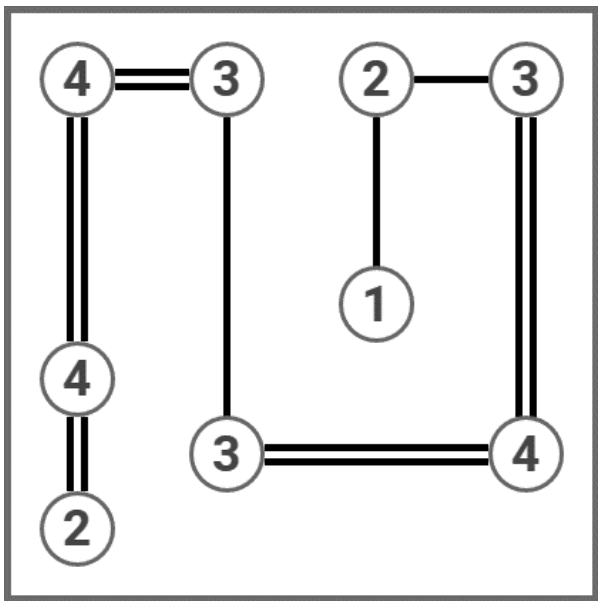
1.1. Zadatak diplomskog rada

Potrebno je izraditi web aplikaciju koja uspješno rješava *Hashiwokakero* zagonetku te omogućuje skeniranje zagonetke uz pomoć kamere. Aplikacija mora omogućiti i ručni unos zagonetke ukoliko korisnik nije u stanju uslikati te riješiti predanu zagonetku i prikazati rješenje.

2. ZAGONETKA HASHIWOKAKERO (MOSTOVI)

Zagonetka mostovi (jpn. *Hashiwokakero*, skraćeno *Hashi*) je logička zagonetka u kojoj je potrebno otoke spojiti s mostovima tako da na kraju svi otoci budu povezani u jednu cjelinu. Zagonetka je prvi put predstavljena javnosti u 31. izdanju magazina *Puzzle Communication Nikoli* u rujnu 1990. godine [1]. Taj isti magazin je proslavio i mnoge druge logičke zagonetke kao što su Sudoku i Kakuro.

U zagonetci *Hashi* igrač je kralj otoka i njegova je zadaća spojiti sve otoke u ovisnosti o njihovoj populaciji. Mali otoci se spajaju sa samo jednim mostom (broj 1 piše na otoku), a veći otoci spajaju se s više mostova (broj mostova piše na otoku). Igrač mora zapamtiti da su svi otoci dio njegovog kraljevstva i zato mora omogućiti svakom građaninu kraljevstva nesmetan put od jednog do drugog otoka [2]. Slika 2.1. prikazuje riješenu zagonetku mostovi iz koje je vidljivo kako svi otoci u konačnici čine jednu cjelinu i kako je svaki otok povezan s drugim otocima s onolikim brojem mostova koji pišu na njemu. Zagonetka može biti bilo koje veličine, a težina rješavanja zagonetke se većinom povećava s njezinom veličinom.



Slika 2.1. Rješenje zagonetke mostovi veličine 7x7 (hashiwokakero).

2.1. Pravila zagonetke

Postoji nekoliko pravila koja opisuju kako se rješava ova zagonetka [2] [3]:

1. Spoji otoke s onoliko mostova koliko piše na otoku
2. Ne može postojati više od dva mosta između dva otoka
3. Mostovi se ne mogu protezati dijagonalno
4. Mostovi se ne mogu protezati preko otoka ili drugih mostova
5. Mostovi moraju na kraju formirati jednu zajedničku grupu (cjelinu)

2.2. Metode rješavanja

Rješavanje zagonetke mostovi svodi se na proceduralnu metodu tako što se postupno rješavaju otoci te dodavanjem novih mostova otvaraju se nove mogućnosti (rješenja) kako postaviti druge mostove. Neke od metoda koje se koriste prilikom rješavanje zagonetke su:

- Otoci koji imaju broj 3 i nalaze se u kutu, otoci koji imaju broj 5 i nalaze se na rubovima te otoci koji imaju broj 7 i nalaze se bilo gdje na polju – za sve te otoke je sigurno da imaju barem jedan most prema svim svojim susjedima
- Otoci koji imaju broj 4 i nalaze se u kutu, otoci koji imaju broj 6 i nalaze se na rubovima te otoci koji imaju broj 8 i nalaze se bilo gdje na polju – za sve te otoke je sigurno da imaju dva mosta prema svim svojim susjedima
- Otoci koji imaju broj 3 i imaju dva susjeda u istoj direkciji (oba otoka su ili vertikalno ili horizontalno pozicionirana od promatranog otoka) –sigurno imaju po jedan most prema svakom susjednom otoku
- Otoci koji imaju broj 5 i nalaze se bilo gdje na polju te imaju tri susjeda –sigurno po jedan most prema svakom otoku
- Potrebno je paziti da se spajanjem otoka ne stvori struktura koja je zatvorena i izolirana, a da to nije konačna cjelina (npr. spajanje dva otoka s brojem 1 je dozvoljeno samo ako su to jedini otoci na polju)
- Prilikom rješavanja je uobičajeno označiti završene otoke kao gotovim

3. PROGRAMSKO RJEŠENJE WEB APLIKACIJE

U ovom poglavlju su opisane sve osnovne tehnologije, programski jezici, okviri i biblioteke koje su korištene pri izradi aplikacije. Također, opisan je algoritam rješavanja zagonetke te algoritam prepoznavanja zagonetke iz predane slike. Na kraju je prikazana implementacija korisničkog sučelja aplikacije i spajanja svih dijelova u jednu funkcionalnu cjelinu.

3.1. Korištene tehnologije i alati

3.1.1. Programski jezici

HTML – (engl. *HyperText Markup Language*) je prezentacijski jezik koji služi za prikazivanje korisničkog sučelja na web stranicama. On je kamen temeljac svake web stranice jer se kroz njega definira smisao i struktura sadržaja. Pod pojmom hipertekst se podrazumijevaju veze koje povezuju sadržaj unutar jedne stranice ili između više stranica. HTML je baziran na XML jeziku kojeg odlikuju „elementi“ koji daju smisao sadržaju. Neki od tih blokova su: <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, … Uz HTML se još najčešće spominju CSS (omogućuje stiliziranje web stranica) i JavaScript (omogućuje funkcionalnost i interaktivnost web stranicama) kao osnovni jezici koji pokreću web stranice [4].

JavaScript – programski jezik koji je najbolje poznat po tome što se koristi kao jezik koji omogućuje interakciju s web stranicama. JavaScript je dinamičan interpretacijski jezik i spada u razne paradigme jezika te ga je kao takvog teško svrstati u jednu strukturu. Nudi mogućnost objektnog, funkcionalnog i imperativnog programiranja, a baziran je na prototipovima i funkcijama kao osnovnom strukturom jezika. Ovaj jezik iako je osmišljen samo za rad s web stranicama, danas se ipak koristi i u raznim drugim okruženjima kao što su poslužitelji i mobilne aplikacije. Prva verzija JavaScript jezika stvorena je od strane Brendan Eicha u Netscapeu te je od samog početka jezik osmišljen da bude što jednostavniji za naučiti. Tako su u jezik ubačeni poznati programski oblici i metode iz jezika kao što su C++ i Java [5].

Ono što JavaScript čini posebnim je to što je danas dostupan svugdje, pa je zato za ovaj diplomski rad izabran kao glavni jezik. S njim će biti omogućeno stvoriti web aplikaciju koja će se moći otvoriti u svim Internet preglednicima (i na stolnom računalu i na mobilnom telefonu) te pronaći

rješenje zagonetke bez oslanjanja na bilo kakav poslužitelj. Tako je sva logika prebačena na klijentsku stranu, od prikazivanja web aplikacije, preko detekcije zagonetke preko kamere (slike) do rješavanja samo zagonetke.

TypeScript – programski jezik otvorenog koda nastao 2012. godine, a koji je baziran na programskom jeziku JavaScript i razvijan od strane Microsofta [6]. Ovaj jezik na JavaScript dodaje opcionalne statične tipove kojima se opisuju varijable i objekti. Procesom prevođenja svaki TypeScript kod se prevodi nazad u JavaScript. Prednosti koje TypeScript nudi u odnosu na JavaScript su bolji alati za razvojne inženjere koji omogućuju lakšu promjenu koda te provjeru grešaka tijekom postupka prevođenja [7].

CSS – (engl. *Cascading Style Sheets)* jezik kojim opisujemo kako će elementi web stranice izgledati. Daje nam razne mogućnosti stiliziranja od jednostavnih stvari kao što su mijenjanje veličine i boje teksta, preko definiranja pozicije elemenata do kompleksnih stvari kao što je animacija elemenata na web stranici. CSS opisuje kako će elementi biti prikazani na zaslonu, na papiru, u govoru ili drugim medijima [8].

WebAssembly - jezik sličan strojnom jeziku (on je tip koda koji se može izvršavati unutar web preglednika) niske razine s kompaktnim binarnim formatom koji se izvršava skoro brzo kao i izvorni kod, a omogućuje jezicima kao što su *C / C* ++ i *Rust* da se kompiliraju u njega i tako omoguće izvršavanje unutar web preglednika. Također je dizajniran da se pokreće uz JavaScript, što omogućuje da oba jezika rade zajedno [9].

WebAssembly ima ogromne implikacije za web platformu - pruža način za pokretanje koda pisanog na više jezika na webu u blizini izvorne brzine, s klijentskim aplikacijama koje se pokreću na webu, a koje prethodno to nisu mogle učiniti.

WebAssembly je osmišljen kako bi nadopunio JavaScript i bio pokretan zajedno s njim - koristeći WebAssembly JavaScript API-je, može se učitati *WebAssembly* modul u JavaScript aplikaciju i podijeliti funkcionalnost. To omogućuje iskorištavanje prednosti WebAssembly-a te izražajnost i fleksibilnost JavaScripta u istim aplikacijama, iako razvojni programer ne zna pisati WebAssembly kod.

3.1.2. Programski okviri i biblioteke

Angular – JavaScript programski okvir koji omogućuje jednostavnu izradu aplikacija baziranih na web tehnologijama. Angular je nastao 2015. godine kao potpuni redizajn postojećeg *AngularJS* programskog okvira. On je programski okvir otvorenog koda koji je aktivno održavan od Googlea [10].

Angular je JavaScript (točnije TypeScript) programski okvir koji omogućuje stvaranje reaktivnih aplikacija s jednom stranicom (engl. *Single Page Applications*). Komponente su glavni dio svake Angular aplikacije. Ugnježđivanjem jedne komponente u drugu dobiva se stablo komponenti aplikacije s točno određenim srodstvom.

Neke od prednosti zbog kojih se Angular koristi su [11]:

- Angular ne pruža samo alate već i primjere kako napraviti projekt kojeg je lako održavati. Kada se Angular projekt uspostavi na ispravan način, ne postoje metode i klase koje je teško promijeniti i testirati.
- Angular je napravljen uz pomoć TypeScripta koji pruža najnovije mogućnosti JavaScripta uz statično određivanja tipova varijabli.
- Postojanje unaprijed definiranih skupova alata koji omogućuju brzo uspostavljanje projekta poput manipuliranja formama i pozivanja REST usluga.
- Sve komponente su odvojene jedne od druge te ih je lako zamijeniti s drugom implementacijom
- Testiranje je moguće na razini komponenti, ali i na razini cijele aplikacije
- Mogućnost izrade mobilnih aplikacija

Slika 3.1. predstavlja logički kod jedne Angular komponente koja omogućuje dodavanje stavki za obavljanje na listu, dok se prezentacijski dio komponente nalazi unutar datoteke za predložak. Svaka komponenta je obična JavaScript klasa na koju je dodan @Component dekorater.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-todos',
  templateUrl: 'template.html',
  styleUrls: ['todos.css']
})

export class TodosComponent {
  newTodo: string;
  todos: any;
  todoObj: any;

  constructor() {
    this.newTodo = '';
    this.todos = [];
  }

  addTodo(event) {
    this.todoObj = {
      newTodo: this.newTodo,
      completed: false
    };
    this.todos.push(this.todoObj);
    this.newTodo = '';
  }
}
```

Slika 3.1. Primjer Angular komponente koja predstavlja listu stvari koje treba obaviti.

OpenCV.js – preinaka popularne OpenCV biblioteke za JavaScript. OpenCV (engl. *Open Source Computer Vision Library*) je biblioteka otvorenog koda za računalni vid i strojno učenje. OpenCV je napravljen kako bi ponudio jedinstvenu i široko prihvatljivu infrastrukturu za izradu aplikacija koje koriste računalni vid (za komercijalne i nekomercijalne svrhe). Biblioteka ima više od 2500 optimiziranih algoritama, među kojima se nalaze klasični algoritmi računalnog vida i moderni algoritmi koji se oslanjaju na strojno učenje. Ti algoritmi se koriste za detekciju i prepoznavanje lica, detekciju objekata, klasificiranje ljudskih pokreta u video sadržaju, praćenje kretanje kamere, stvaranje 3D objekata iz stereo slika, stvaranje visoko-rezolucijske slike od više slika, uklanjanje crvenih očiju iz slika… OpenCV je pisan u C++ jeziku, a može se koristi i iz drugih jezika kao što su Python, Java, C# i JavaScript [12].

OpenCV.js je određeni skup funkcija iz originalne OpenCV biblioteke koje se mogu pozivati iz JavaScript jezika [13]. Omogućuje web aplikacijama širok raspon već gotovih funkcija za obradu slike i videa. OpenCV.js se oslanja na Emscripten alat [14] za kompiliranje originalne OpenCV biblioteke pisane u C++ jeziku u WebAssembly i omogući JavaScript API kojim će se pozivati OpenCV funkcije.

Tesseract.js – JavaScript biblioteka koja može prepoznati znakove iz slika. Nastala je kompiliranjem Tesseract OCR engina (engl. *Optical Character Recognition*) preko Emscripten alata i tako je postao dostupan na web platformi [15].

Tesseract *engine* [16] je originalno bio razvijen u Hewlett Packard između 1985. i 1994. godine, a dodane su još neke sitne promjene u narednim godinama. 2005. godine tvrtka Hewlett Packard odlučuje Tesseract objaviti kao *engine* otvorenog koda, a od 2006. godine Google je glavni sponzor daljnjeg razvoja *engina*. Trenutna verzija do koje je došao razvoj je verzija 4.

Inicijalna verzija je mogla prepoznati samo tekst engleskog jezika, a danas *engine* podržava čak 116 jezika, među kojima je i hrvatski, ali i ideografski jezici poput japanskog i kineskog te jezici koji se pišu s desna na lijevo poput arapskog. Osim toga, moguće je dodatno istrenirati *engine* za potpuno nove jezike.

Rezultat koji će izbaciti Tesseract *engine* nakon prepoznavanja znakova u velikom dijelu ovisi o tome koliko je dobro obavljeno pretprocesiranje slike. Naime, potrebno je sliku obraditi na takav način da se iz slike ukloni šum, znakovi ne budu premale veličine (*engine* očekuje da visina znakova bude barem 20 piksela), rotacija i izobličenost slike mora biti uklonjena te tamni rubovi moraju biti uklonjeni jer bi se mogli prepoznati kao znakovi.

Nakon postupaka pretprocesiranja, obrađena slika se predaje na prepoznavanje znakova. Točnost rezultata obrade može se povećati određivanjem dozvoljenih ili očekivanih znakova. Tako npr. u slučaju ovog diplomskog rada predani su samo brojevni znakovi od 1 do 8 jer nam drugi znakovi nisu potrebni. Tesseract *engine* kao rezultat vraća tekst u kojem pišu svi znakovi koje je uspio pronaći na slici zajedno s preciznošću koliko je siguran da je to taj pronađeni znak.

3.2. Implementacija algoritma za rješavanje zagonetke

3.2.1. Strukture podataka

Korektno definirana struktura podataka omogućuje potpunu transformaciju zagonetke iz fizičkog svijeta u digitalni. Transformacija je nužna kako bi računalo moglo prepoznati i riješiti zagonetku. U ovom diplomskom radu korištene su strukture podataka u obliku matrice te u obliku grafa. Kao ulazni parametar rješavaču predaje se dvodimenzionalna cjelobrojna matrica gdje nule označavaju

prazno polje u fizičkoj zagonetci, a brojevi od jedan do osam označavaju otoke. Slika 3.2. prikazuje primjer dvodimenzionalne matrice koja je implementirana u JavaScript jeziku. Također, matrica ne smije sadržavati nedozvoljene znakove poput slova i interpunkcijskih znakova.

```
[
  [0, 2, 0, 5, 0, 0, 2],
  [0, 0, 0, 0, 0, 0, 0],
  [4, 0, 2, 0, 2, 0, 4],
  [0, 0, 0, 0, 0, 0, 0],
  [0, 1, 0, 5, 0, 2, 0],
  [0, 0, 0, 0, 0, 0, 0],
  [4, 0, 0, 0, 0, 0, 3]
],
```

Slika 3.2. Primjer ulazne dvodimenzionalne matrice algoritma za rješavanje zagonetke.

Izlazna vrijednost rješavača zagonetke ima sličnu strukturu kao i ulazna struktura. Razlikuje se u tome što su na nekim mjestima nule zamijenjene znakovima koji predstavljaju mostove. Ti znakovi su: - (jednostruki horizontalni most), = (dvostruki horizontalni most), | (jednostruki vertikalni most) i $ (dvostruki vertikalni most).

Struktura u obliku matrice je pogodna za promatranje zagonetke kroz prostor jer nam daje jasnu sliku o susjedstvu pojedinih otoka. Osim toga, pogodna je i za prikaz jer se lako svaki pojedini element matrice zamijeni sa slikovnom reprezentacijom. Slika 3.3. prikazuje dvodimenzionalnu matricu koju algoritam rješavač zagonetke vraća kao konačno rješenje. Ovakva struktura ispisa omogućuje jednostavno iščitavanje susjedstva otoka. Iteracijom kroz otoke i dodavanjem mostova u matricu stvaraju se granice te se tako sužava broj mogućih susjeda prema kojima je moguće ostvariti vezu. U konačnici, ulazna matrica će biti potpuno riješena i spremna za prikaz.

```
[
  [ 0 ,  2 , '=',  5 , '-', '-',  2 ],
  [ 0 ,  0 ,  0 , '$',  0 ,  0 , '|'],
  [ 4 , '=',  2 , '$',  2 , '=',  4 ],
  ['$',  0 ,  0 , '$',  0 ,  0 , '|'],
  ['$',  1 , '-',  5 , '=',  2 , '|'],
  ['$',  0 ,  0 ,  0 ,  0 ,  0 , '|'],
  [ 4 , '=', '=', '=', '=', '=',  3 ]
],
```

Slika 3.3. Primjer izlazne vrijednosti rješavača zagonetke.

Iako je dvodimenzionalna matrica najbolji izbor za ulazne i izlazne vrijednosti, ona ipak nije pogodna za efektivan algoritam rješavanja. Glavni nedostatak ove strukture je što nije lako pronaći susjede te provjeriti postojanje samo jedne zajedničke cjeline (otoci moraju biti svi spojeni u jednu cjelinu). Zbog toga, uvedena je struktura u obliku grafa kako bi nadopunila nedostatke koje ima struktura matrice. Algoritam za rješavanje koristi i jednu i drugu strukturu u ovisnosti o potrebama.

U strukturi grafa postoje čvorovi i veze, gdje čvorovi predstavljaju otoke s njihovim podatcima, a veze predstavljaju mostove. Svaki čvor sadrži svoj identifikacijski broj, vrijednost otoka, poziciju otoka, veze prema ostalim otocima i stanje završenosti (otok je završen ako ima sve potrebne mostove). Otok može imati maksimalno četiri susjeda pa tako svaki čvor ima i četiri veze. Ukoliko neka veza nije moguća, onda ta veza nema vrijednost. Svaka veza osim reference na susjedni otok, nosi i informacije o broju povučenih mostova, poziciji susjednog otoka i stanju završenosti. Slika 3.4. prikazuje opisanu strukturu koja je implementirana unutar algoritma za rješavanje zagonetke.

```
interface GraphNode {
  id: number;
  completed: boolean;
  position: number[];
  value: number;
  neighbors: Neighbor[];
}

interface Neighbor {
  bridges: number;
  position: number[];
  done: boolean;
  node: GraphNode;
}
```

Slika 3.4. Struktura grafa korištena unutar algoritma za rješavanje zagonetke.

3.2.2. Inicijalizacija rješavača

Proces inicijalizacije rješavača svodi se na pripremu strukture u obliku grafa tj. transformaciju iz dvodimenzionalne matrice u graf. Prilikom inicijalizacije kroz matricu pronalaze se susjedi te popunjava graf koristeći implementirane *traverse* pomoćne funkcije (*traverseUp, traverseRight, traverseDown, traverseLeft*).

Funkcija *traverseUp* će, u odnosu na zadanu početnu točku, pretražiti matricu prema gore te ukoliko pronađe otok vratiti njegovu poziciju zajedno s vrijednošću. Ukoliko nije pronađen nijedan otok ili je prilikom pretrage pronađen most, funkcija tada vraća nulu. Slika 3.5. prikazuje implementaciju navedene *traverseUp* funkcije (na sličan način su implementirane ostale funkcije za pronalazak otoka prema desno, dolje i lijevo).

```
function traverseUp(x: number, y: number) {
  for (let row = x - 1; row ≥ 0; row--) {
    if (puzzle[row][y] === '=' || puzzle[row][y] === '-') {
      return 0;
    }
    if (puzzle[row][y] > 0) {
      return { position: [row, y], value: puzzle[row][y] };
    }
  }
  return 0;
}
```

Slika 3.5. TraverseUp funkcija za pronalazak mostova i otoka iz matrice.

Nakon završenog postupka inicijalizacije dobiven je inicijalni graf s određenim susjedstvima gdje sve veze imaju broj mostova 0 i stanje završenosti. Time je graf spreman i šalje se algoritmu za rješavanje koji će iterativno riješiti zagonetku.

3.2.3. Tijek rješavanja zagonetke

Rješavanje zagonetke odvija se iterativno u više koraka. Svakim novim korakom dodaju se novi mostovi sve dok zagonetka nije u potpunosti riješena. Broj koraka kroz koje će zagonetka biti riješena nije unaprijed poznat već je dinamičan i ovisi o određenim uvjetima. Na kraju svakog koraka provjerava se riješenost zagonetke, a zagonetka je riješena ako svi otoci imaju stanje završenosti postavljeno na istinitu vrijednost i ako zagonetka čini jednu cjelinu tj. broj čvorova u grafu koji počinje od prvog čvora mora biti jednak sveukupnom broju čvorova (Slika 3.6.). Osim toga, ako u jednom koraku rješavanja nije dodan niti jedan most, onda se zagonetka prekida i nije uspješno riješena. Ovaj uvjet ostvaruje se uspoređivanjem trenutnog stanja zagonetke sa stanjem koje je bilo u prošlom koraku. Ukoliko su ta dva stanja jednaka, prekida se rješavanje.

```
if (
  islands === completedIslands &&
  numberOfGraphNodes(graphNodes[0]).length === islands
) {
  console.log('--------SOLVED---------');
  return { solved: true, puzzle };
}
```

Slika 3.6. Provjera riješenosti zagonetke.

U svakom koraku rješavanja algoritam prolazi kroz sve otoke koji nisu završeni i za njih provjerava mogućnost dodavanja mostova. Prije samog početka provjere postavljaju se varijable stanja otoka koje opisuju trenutno stanje otoka i njegove okoline (Slika 3.7.):

- count – broj susjednih otoka koji nisu gotovi
- completed – broj mostova prema susjednim otocima koji su gotovi
- sum – broj mostova prema susjednim otocima
- incomplete – broj mostova prema susjednim otocima kojima veza nije gotova
- neighboringValues – zbroj vrijednosti susjednih otoka prema kojima veza nije gotova

```
const count = gn.neighbors.filter(n => n !== null && !n.node.completed)
  .length;
const completed = gn.neighbors
  .filter(n => n !== null && n.node.completed)
  .map(n => n.bridges)
  .reduce((a, b) => a + b, 0);
const sum = gn.neighbors
  .filter(n => n !== null)
  .map(n => n.bridges)
  .reduce((a, b) => a + b, 0);
const incomplete = gn.neighbors
  .filter(n => n !== null && n.done === false)
  .map(n => n.bridges)
  .reduce((a, b) => a + b, 0);
const neigboringValues = gn.neighbors
  .filter(n => n !== null && !n.done)
  .map(v => v.node.value)
  .reduce((a, b) => a + b, 0);
```

Slika 3.7. Postavljanje pomoćnih varijabli.

Nakon uspostavljenih varijabli stanja otoka prolazi se kroz dio algoritma zaslužan za provjeru stanja otoka. Neki od mogućih uvjeta kada se povlači most između dva otoka su:

- ako oko otoka postoje otoci s vrijednostima jedan i dva, onda dodijeli jedan most prema otoku s brojem dva
- ako otok ima samo jednog susjeda, onda dodijeli jedan ili dva mosta prema tom otoku
- ako otok ima vrijednost 3 i dva susjeda, onda dodijeli po jedan most prema susjedima (isto vrijedi i za otok s brojem 5 s tri susjeda te otok s brojem 7 s četiri susjeda)
- ako otok ima vrijednost 4 i dva susjeda, onda dodijeli po dva mosta prema susjedima (isto vrijedi i za otok s brojem 6 s tri susjeda te otok s brojem 8 s četiri susjeda)

Slika 3.8. prikazuje skraćeni kod u kojem se nalaze svi uvjeti za koje se povlači jedan ili dva mosta od jednog otoka prema drugom.

```
if (count === 2 && neigboringValues === 3 && sum === 0) {…
} else if (count === 1) {…
} else if (count === 2 && gn.value - sum + incomplete > 2) {…
} else if (
  count === 2 &&
  gn.value === 2 &&
  sum === 0 &&
  (neigboringValues === 3 || neigboringValues === 4)
) {…
} else if (
  gn.value === 2 &&
  count === 2 &&
  sum === 0 &&
  neigboringValues > 4 &&
  gn.neighbors.filter(n => n !== null && n.node.value === 1).length ===
    1
) {…
} else if (count === 2 && gn.value === 1) {…
} else if (count === 2 && gn.value - sum + incomplete === 2) {…
} else if (
  ((gn.value === 5 || gn.value === 6) &&
    count === 3 &&
    count === gn.neighbors.filter(n => n !== null).length) ||
  ((gn.value === 7 || gn.value === 8) && count === 4)
) {…
} else if (gn.value === 5 && sum === 3 && count === 2) {…
} else if (
  (count === 3 && gn.value === 3) ||
  (count === 4 && gn.value === 4) ||
  (count === 4 && gn.value === 6)
) {…
} else if (gn.value === 4 && count === 3 && gn.value - sum > 2) {…
} else if (gn.value === 5 && count === 3 && gn.value - sum > 2) {…
} else if (gn.value === 6 && count === 3 && sum === 2) {…
} else if (gn.value === 6 && completed === 1 && count === 3) {…
}
```

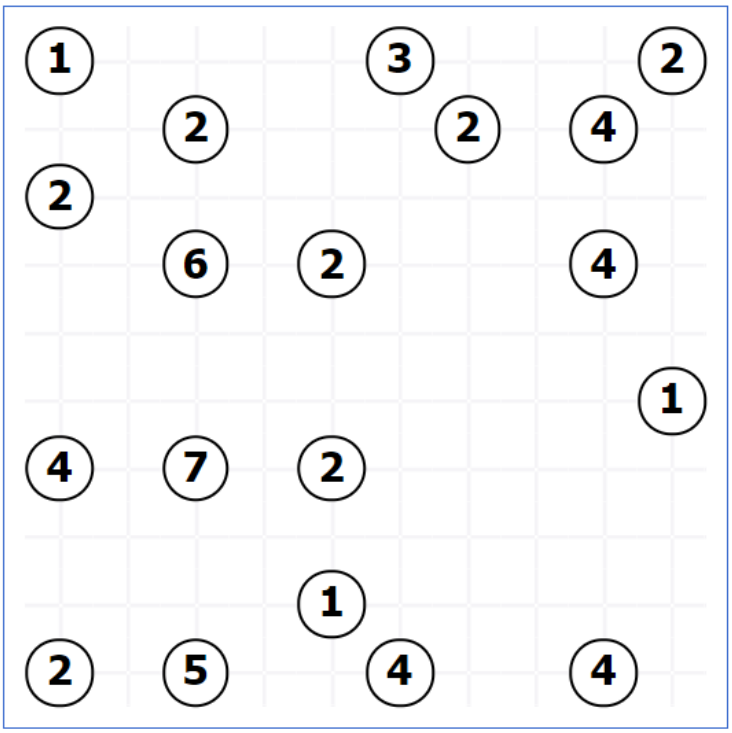
Slika 3.8. Svi uvjeti za koje se povlače mostovi između otoka.

Prilikom svakog dodjeljivanja mostova između otoka koristi se funkcija *setBridges* koja prima otok od kojega se povlači most, otok do kojeg se povlači most, broj mostova i smjer mosta. Ova funkcija će onda unutar grafa postaviti veze od prvog otoka prema drugom i obrnuto te ako je otok ili samo veza gotova – promijeniti stanje u završeno.

Nakon provjerenih svih uvjeta, u odnosu na nove veze koje su uspostavljene unutar grafa, ažurira se matrična reprezentacija zagonetke i čiste sve veze iz grafa koje više ne postoje. Ako je neki od otoka postao završen, onda se uklanja njegova vidljivost prema susjednim otocima s kojim nema vezu. Također, ako je povučena veza između dva otoka i tako stvorena granica između određenih otoka, ti otoci nakon postupka čišćenja neće moći više vidjeti jedan drugoga.

3.3. Detekcija zagonetke iz slike

Detekcija zagonetke mostovi izvršava se u dva koraka. Prvi korak je detekcija pozicije otoka, a drugi korak je prepoznavanje broja koji se nalazi na otoku. U ovom poglavlju će biti objašnjena ta dva koraka. Slika 3.9. prikazuje zagonetku nad kojom će se u ovom poglavlju prezentirati primjer detekcije zagonetke iz slike.

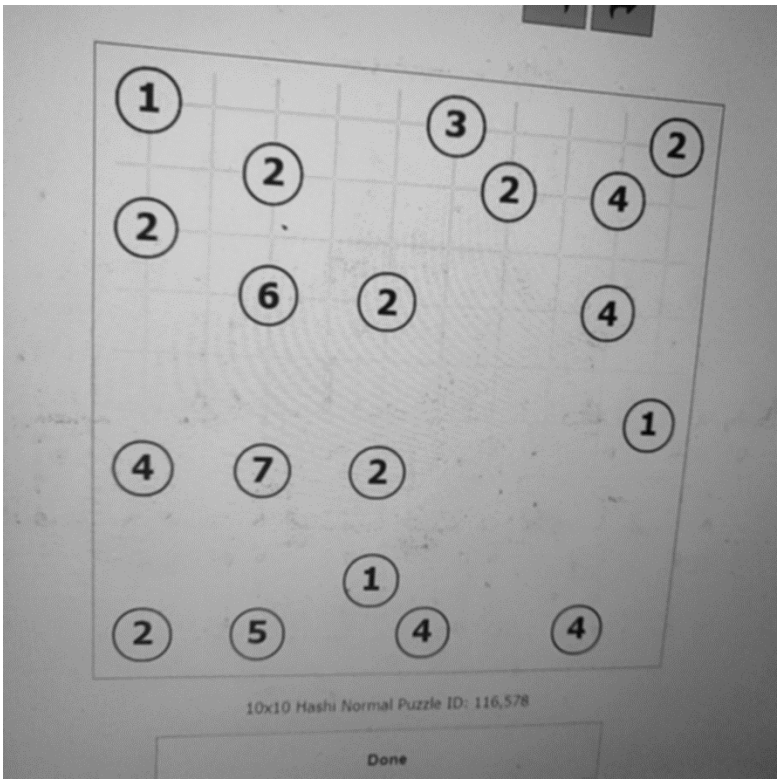


Slika 3.9. Zagonetka nad kojom će se vršiti primjer algoritma detekcije zagonetke iz slike.

3.3.1. Detekcija pozicija otoka

Detekcija pozicije otoka se svodi na prepoznavanje krugova koji se nalaze na polju. Kroz cijeli algoritam se proteže pretpostavka da je zagonetka u standardnom obliku tj. bijelo pravokutno polje s crnim obrubom unutar kojeg se nalaze kružnice crne boje unutar kojih se nalaze brojevi crne boje. Detekcija pozicije se u potpunosti izvodi uz pomoć OpenCV.js biblioteke koja ima potrebne algoritme za manipuliranje značajkama slike.

Slika 3.10. prikazuje sliku zagonetke koja je uslikana pomoću mobilnog uređaja i ona predstavlja najčešći uzorak koji dobiva algoritam za procesiranje slike. Naime, slika je uslikana pod kutom tako da je došlo do iskrivljenja cijele zagonetke (dijelovi slike koji su bili bliži kameri tijekom slikanja izgledaju uvećano, a dijelovi koji su bili udaljeniji izgledaju umanjeno). Iskrivljenje je potrebno ispraviti kako bi se pravilno mogle odrediti dimenzije zagonetke i pozicije pojedinih otoka.



Slika 3.10. Uslikana zagonetka pomoću mobilnog uređaja.

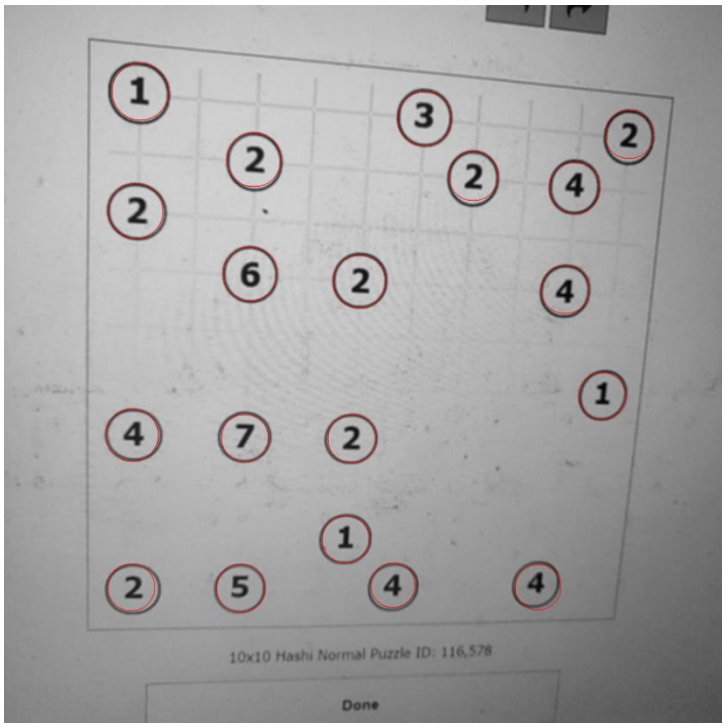
Prvi korak koji je potrebno napraviti je pretprocesiranje slike u kojem se slika pretvara u crno-bijelu sliku i nakon toga se primjenjuje Gaussovo zamućenje nad cijelom slikom. Pretvaranje slike u crno-bijelu uklanja nepotrebne boje iz slike jer detekcija znakova radi najbolje nad crno-bijelim slikama. Zamućivanje slike omogućuje bolju detekciju krugova, a u isto vrijeme se iz slike uklanja neželjeni šum ili neki drugi sitni artefakti unutar slike.

Nakon što je slika zamućena primjenjuje se algoritam Houghove transformacije nad krugovima (engl. *Circle Hough Transform*, skraćeno kao CHT). CHT je jedna od osnovnih tehnika u obradi slike za detekciju krugova unutar slike. CHT je specijalizacija generalne Houghove transformacije, a svrha algoritma je pronaći krugove u slikama s lošom kvalitetom (krugovi mogu biti isprekidani i nepotpuni). Detekcija se izvršava pozivanjem funkcije *HoughCircles* iz OpenCV biblioteke koja kao rezultat vraća sve krugove koje je pronašla s njihovom x i y koordinatom središta te radijusom.

Dobivanje pozicija svih otoka unutar zagonetke vrši se u tri koraka:

1. Pronalaženje krugova unutar iskrivljenje slike
2. Uklanjanje iskrivljenja slike
3. Ponovno pronalaženje krugova (ovaj put oni imaju ispravne pozicije)

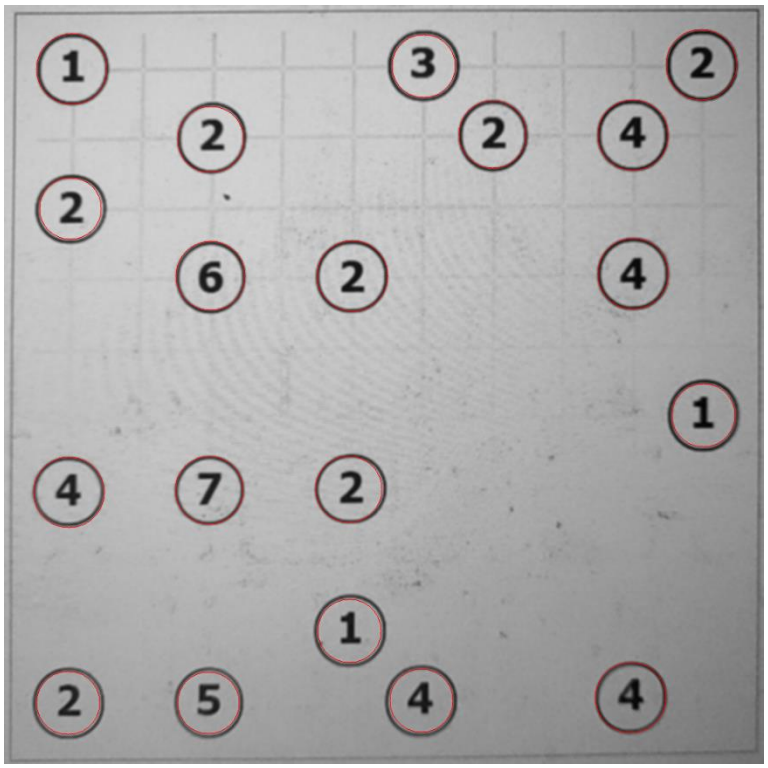
U prvom koraku nakon pronalaska svih krugova (Slika 3.11.) promatraju se po dva kruga koja se nalaze na samim bridovima zagonetke. Pomoću njihovih pozicija određuju se pravci na kojima se oni nalaze te izračunava sjecište pravaca. Traženjem sjecišta gornjeg pravca s desnim pravcem dobiva se gornja desna točka zagonetke (analogno tome određuju se i ostale točke kutova zagonetke). Dobivene kutne točke predaju se sljedećem koraku koji vrši uklanjanje iskrivljenja slike.



Slika 3.11. Detektirani krugovi unutar zagonetke metodom Houghove transformacije (detektirani krugovi su označeni crvenom bojom).

U drugom koraku se od dobivenih kutnih točaka stvara transformacijska matrica i mijenja perspektiva slike kako bi se dobila slika kao da je slikana direktno iznad zagonetke bez ikakve rotacije i iskrivljenja.

U trećem koraku se vrši ponovo detekcija krugova te ovaj put dobivaju se ispravne pozicije svih otoka. Na Slika 3.12. prikazana je dobivena slika bez iskrivljenja te je vidljivo da je prilikom promjene perspektive ujedno i promijenjena veličina slike tako da se u njoj samo nalazi zagonetka.



Slika 3.12 Slika zagonetke nakon što je uklonjeno iskrivljenje i u kojoj su crvenom bojom označeni detektirani krugovi.

Nakon što su detektirani svi krugovi unutar slike koji predstavljaju otoke potrebno je detektirati dimenzije polja kako bi se nakon detekcije znakova moglo polje u obliku matrice poslati rješavaču zagonetke na rješavanje. Postupak određivanja dimenzija polja je isti i za redove i za stupce, a on se ostvaruje na sljedeći način (na isti način se pronalazi broj redova, tako što umjesto x koordinata koristimo y koordinate):

1. Sortiranjem svih x koordinata krugova u listu
2. Određivanjem udaljenosti između otoka koja je postavljena na dvostruku veću odaljenost od radijusa otoka
3. Dijeljenjem najveće x koordinate s pronađenom najmanjom udaljenosti i dodavanjem jedan dobivamo broj stupaca

Slika 3.13. prikazuje kod funkcije koja vraća dimenziju zagonetke tj. broj redaka ili stupaca. Osim dimenzija, ova funkcija vraća i udaljenost između otoka te poziciju prvog otoka koja je potrebna u daljnjem pronalasku pozicije otoka. Naime, poznavanje x i y koordinate ne daje velik značaj rješavanju zagonetke pa ih zato moramo transformirati u koordinate stupaca i redaka. Prema (3-1) oduzimanjem trenutne x vrijednost od x vrijednosti prvog otoka i dijeljenjem njihove razlike s

veličinom udaljenosti dobiva se broj stupca kojem otok pripada. Ista logika se primjenjuje i za određivanja retka tako što se uvrste y koordinate umjesto x koordinata.

$$stupac = \left\lfloor \frac{|x - x_{prvi}|}{udaljenost} \right\rfloor \tag{3-1}$$

$$redak = \left\lfloor \frac{|y - y_{prvi}|}{udaljenost} \right\rfloor \tag{3-2}$$

```
function getDimension(coords: any[], radius: number) {
  const positions = coords.map(x => Math.round(x)).sort((a, b) => a - b);
  const distances = [];
  positions.reduce((a, b) => {
    distances.push(b - a);
    return b;
  });
  const positionBins = createBins(positions);
  const firstPosition = Math.min( ... positions);
  const distance = radius * 2;
  const dimension =
    Math.floor(
      (Math.max( ... positionBins) - Math.min( ... positionBins)) / distance
    ) + 1;
  return [dimension, distance, firstPosition];
}
```

Slika 3.13. Funkcija koja vraća dimenzije zagonetke.

Slika 3.14. prikazuje dio koda koji je zaslužan za izrezivanje brojeva iz zagonetke. Kako bi OCR *engine* mogao efektivno pronaći broj koji se nalazi unutar kruga, dodatno se vrši ekstrakcija tog broja na takav način da prvo obojimo krug koji je crne boje u bijelu (krug tako nestaje) te izrežemo područje oko broja. Potom sve pronađene brojeve spajamo u jednu sliku i nju šaljemo dalje na prepoznavanje znakova. Slika 3.15. prikazuje brojeve koji su pronađeni iz originalne slike i koji će biti dalje proslijeđen OCR *enginu* na prepoznavanje.

```
const numberPositions = [];
const detectedImages = new cv.MatVector();
for (const coord of coords) {
  const x = coord.x;
  const y = coord.y;
  const center = new cv.Point(x, y);

  const row = Math.floor(Math.abs(y - firstY) / yDistance);
  const column = Math.floor(Math.abs(x - firstX) / xDistance);

  cv.circle(dst, center, maxradius, [255, 255, 255, 255], maxradius * 0.5);

  // get portion of image with number
  const rect = new cv.Rect(
    x - maxradius,
    y - maxradius,
    2 * maxradius,
    2 * maxradius
  );
  const numberImage = dst.roi(rect);

  cv.threshold(numberImage, numberImage, 127, 255, cv.THRESH_BINARY);
  detectedImages.push_back(numberImage);

  numberPositions.push({ row, column });
}
```

Slika 3.14. Kod izrezivanja broja iz slike.

Slika 3.15. Prepoznati brojevi iz zagonetke koji su spojeni u jednu cjelinu.

3.3.2. Prepoznavanje broja na otoku

Prepoznavanje znakova se vrši uz pomoć Tesseract.js biblioteke koja u pozadini koristi Tesseract OCR *engine*. Tesseract.js biblioteka zahtijeva postavljanje parametara koji opisuju prepoznavane znakove. U ovom diplomskom radu postavljeno je prepoznavanje znakova od 1 do 8, jer nas samo ti znakovi zanimaju, a kao jezik postavljen je engleski (jezik je mogao biti i neki drugi, samo je važno postojanje arapskih brojeva u tom jeziku). Slika 3.16. prikazuje kod koji iz slike s brojevima prepoznaje te brojeve i obavještava korisnika o trenutnom progresu detektiranja znakova.

```
const tesseractResult = await worker
  .recognize(allNumbersCanvas, 'eng', {
    tessedit_ocr_engine_mode: OEM.TESSERACT_ONLY,
    tessedit_char_whitelist: '12345678'
  })
  .progress(message => {
    notifier.next(message.status);
    console.log(message);
  });
```

Slika 3.16. *Prepoznavanje brojeva iz slike.*

3.4. Implementacija korisničkog sučelja

3.4.1. Struktura aplikacije

Početna točka aplikacije je *main.ts* datoteka u kojoj se vrši pokretanje Angular programskog okvira tako što se odabire početni *NgModule*. *NgModule* opisuje kako se dijelovi aplikacije međusobno uklapaju. Svaka aplikacija ima barem jedan Angular modul tj. korijenski modul koji se pokreće kako bi se pokrenula sama aplikacija. Po dogovoru obično se naziva *AppModule* (Slika 3.17.).

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ServiceWorkerModule } from '@angular/service-worker';
import { environment } from '../environments/environment';
import { SolutionComponent } from './solution/solution.component';
import { HomeComponent } from './home/home.component';
import { PuzzleInputComponent } from './puzzle-input/puzzle-input.component';
import { ReactiveFormsModule } from '@angular/forms';
import { HeaderComponent } from './header/header.component';

@NgModule({
  declarations: [
    AppComponent,
    SolutionComponent,
    HomeComponent,
    PuzzleInputComponent,
    HeaderComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    ServiceWorkerModule.register('ngsw-worker.js', {
      enabled: environment.production
    })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Slika 3.17. Početni modul Angular aplikacije.

Kako bi neka klasa postala Angular modul na nju se dodaje *@NgModule* dekorater. Taj dekorater prima metapodatke koje govore Angularu kako izgraditi i pokrenuti aplikaciju. Neke od vrijednosti koje dekorater može primiti su:

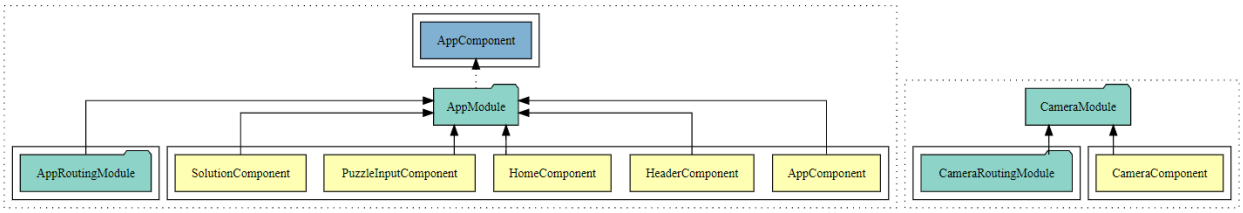
- declarations – polje Angular komponenti koje pripadaju modulu
- imports – polje ostalih uvezenih modula koji se koriste unutar modula
- exports – polje izvoznih komponenti, servisa i modula koje ostali moduli mogu koristiti ako ovaj uvezu ovaj modul
- bootstrap – korijenska komponenta koja se stvara pri pokretanju modula

Osim modula u Angular svijetu, postoje i komponente. Komponente su sve klase koje imaju dekorater *@Component*. Metapodatci za komponentu govore Angularu gdje treba nabaviti glavne građevne blokove koji su potrebni za stvaranje i prezentaciju komponente te njezinog predloška. Metapodatci *@Component* dekoratera koji se najčešće koriste su:

- selector – CSS selektor koji identificira komponentu u predlošku
- temlateUrl – relativna putanja ili apsolutni URL predloška Angular komponente

- styleUrls – polje relativnih putanja ili apsolutnih URL-ova datoteka koje sadrže CSS pravila koja će se primijeniti u komponenti

Povezivanjem komponenti i modula dobiva se hijerarhijsko stablo modula (Slika 3.18.). Ova aplikacija se sastoji od dva modula: *AppModule* i *CameraModule*. Oni su na slici odvojeni jedan od drugoga jer ne ovise jedan o drugome. *AppModule* je početni modul aplikacije, dok *CameraModule* je modul koji se učitava kada se pokrene zaslon za prepoznavanje zagonetke uz pomoć kamere.



Slika 3.18. Hijerarhijsko stablo komponenti i modula aplikacije

AppModule se sastoji od sljedećih komponenti i modula:

- AppComponent
- AppRoutingModule
- HomeComponent
- HeaderComponent
- SolutionComponent
- PuzzleInputComponent

AppRoutingModule je modul koji definira putanje i njihove komponente. Glavni dio ovog modula je *routes* konstanta u kojoj su definirane putanje (Slika 3.19.). Pri praznoj putanji učitat će se *HomeComponent*, dok će putanja *camera* učitati *CameraModule*. Ukoliko korisnik unese putanju koja ne postoji unutar aplikacije, aplikacija će biti preusmjerena na praznu putanju tj. na *HomeComponent*.

```
const routes: Routes = [
  {
    path: '',
    component: HomeComponent
  },
  {
    path: 'camera',
    loadChildren: () ⇒
      import('./camera/camera.module').then(m ⇒ m.CameraModule)
  },
  { path: '**', redirectTo: '' }
];
```

Slika 3.19 Definicija putanja unutar aplikacije.

3.4.2. Komponente

AppComponent je početna komponenta modula. Ova komponenta ne pruža neku veliku funkcionalnost osim što sadrži router-outlet komponentu koja omogućuje da se unutar nje prikažu druge komponente u ovisnosti o trenutnoj putanji.

HomeComponent je komponenta koja je zaslužna za prikaz početnog zaslona na kojem se nalazi polje za unos zagonetke kao i gumbi za rješavanje, poništavanje i slikanje zagonetke uz pomoć kamere (Slika 3.20.). Ona se može podijeliti na 3 dijela. Prvi dio čini samo *HeaderComponent* koji prikazuje tzv. zaglavlje aplikacije koje sadrži logo i ime. Drugi dio čini sama zagonetka koja može biti u dva stanja: riješeno stanje i stanje unosa. Ukoliko je u stanju unosa bit će prikazan *PuzzleInputComponent* s poljem unosa zagonetke i mogućnošću promjene dimenzija zagonetke. Ako je zagonetka riješena, umjesto polja za unos preuzet će *SolutionComponent* i prikazati riješenu zagonetku. Zadnji i treći dio *HomeComponent* zagonetke čine tri gumba: *Solve*, *Camera* i *Restart*. Gumb *Solve* je ljubičaste boje jer je on primarni gumb na zaslonu i kao takav mora biti lako uočljiv korisniku.

```
<app-header></app-header>

<div class="input">
  <app-puzzle-input
    *ngIf="!solved; else solution"
    (statusChanged)="valid = $event"
    [puzzle]="puzzle"
    (puzzleChanged)="puzzleChanged($event)"
  ></app-puzzle-input>
  <div class="not-solved" *ngIf="!!solvedPuzzle && !solved">
    😱 We couldn't solve this puzzle.
  </div>
  <button (click)="solve()" class="button" [disabled]="!valid || empty">
    <img class="icon" src="assets/icons/puzzle.svg" alt="" /><span>SOLVE</span>
  </button>
  <button class="button button--secondary" (click)="openCamera()">
    <img class="icon" src="assets/icons/camera.svg" alt="" /><span>CAMERA</span>
  </button>
  <button (click)="restart()" class="button button--secondary">
    <img class="icon" src="assets/icons/restart.svg" alt="" /><span
      >RESTART</span
    >
  </button>
</div>

<ng-template #solution>
  <div class="solution" *ngIf="solved">
    <app-solution [puzzle]="solvedPuzzle"></app-solution>
  </div>
</ng-template>
```

Slika 3.20. HTML predložak HomeComponent komponente.

Pri svakom učitavanju *HomeComponent* komponente dolazi i do provjere trenutne putanje. Ukoliko putanja sadrži upitni parametar *puzzle* s predanom zagonetkom, onda će se ta zagonetka proslijediti dalje rješavaču zagonetke te biti prikazana.

HeaderComponent je komponenta sa svrhom prikaza imena i loga aplikacije. Kao glavna boja aplikacije izabrana je ljubičasta, pa je zbog toga i HeaderComponent u tom stilu. Odvajanjem loga i imena aplikacije u zasebnu komponentu omogućuje se u daljnjem razvoju aplikacije ponovnu iskoristivost komponente.

SolutionComponent je komponenta za prikaz rješenja zagonetke. Ona ima jedan ulaz (engl. *Input*) preko kojeg prima zagonetku koja se treba prikazati. Prilikom prvog iscrtavanja rješenja zagonetke vidljiva je animacija gdje se svi mostovi protežu od sredine prema otocima koje spajaju, dok otoci povećavaju svoju veličinu i postaju vidljivi. Svaki otok i most, na mobilnim uređajima, imaju veličinu koja je razmjerna širini trenutnog zaslona što znači da će otoci biti manji ukoliko zaslon bude manji. Time je dobivena maksimalna iskoristivost mogućeg prostora. Slika 3.21. prikazuje HTML predložak (engl. *template*) koji definira strukturu komponente.

```
<div>
  <div
    *ngFor="let row of puzzle; let ri = index"
    class="solution__box"
    [style.fontSize.vw]="fontSize"
  >
    <ng-container *ngFor="let c of row; let ci = index" [ngSwitch]="c">
      <span *ngSwitchCase="'|'" class="box">
        <span class="oneVertical"></span>
      </span>
      <span *ngSwitchCase="'$'" class="box">
        <span class="twoVertical"></span>
      </span>
      <span *ngSwitchCase="'-'" class="box">
        <span class="oneHorizontal"></span>
      </span>
      <span *ngSwitchCase="'='" class="box">
        <span class="twoHorizontal"></span>
      </span>
      <span *ngSwitchCase="0" class="box">
        <span></span>
      </span>
      <span *ngSwitchDefault class="box">
        <span class="number">{{ c }}</span>
      </span>
    </ng-container>
  </div>
</div>
```

Slika 3.21. HTML predložak SolutionComponent komponente.

Rješavač zagonetke vraća zagonetku u formatu u kojem su mostovi označeni posebnim znakovima. Zbog toga se mora napraviti mapiranje iz tih znakova u polja zagonetke (Tablica 3.1.) kako bi se prikazali jednostruki i dvostruki mostovi, prazna polja i otoci s njihovim brojevima.

Tablica 3.1. Mapiranje znakova rješenja zagonetke u polja zagonetku.

Znak	Iscrtano polje
-	Jednostruki horizontalni most
=	Dvostruki horizontalni most
|	Jednostruki vertikalni most
$	Dvostruki vertikalni most
0	Prazno polje
Sve ostalo (brojevi)	Otok s brojem u sredini

PuzzleInputComponent je komponenta preko koje se ručno unosi zagonetka. Ova komponenta sadrži jedan ulaz imena *puzzle* preko kojega može primiti neku zagonetku. Ukoliko nijedna zagonetka nije predana, komponenta će biti inicijalizirana s praznom zagonetkom veličine 7x7.

Osim ulaza, komponenta ima i dva izlaza (engl. *Output*). *PuzzleChanged* izlaz će se aktivirati svaki put kada se unos zagonetke promijeni i ostane u validnom stanju (nema nedozvoljenih znakova) te onda poslati trenutno stanje zagonetke nadređenoj komponenti koja prisluškuje događaje ove komponente. Drugi izlaz komponente je *statusChanged* koji se aktivira svaki put kada unesena zagonetka promijeni svoje stanje validnosti. Ovaj izlaz se u nadređenoj komponenti (*HomeComponent*) koristi kako bi se onemogućilo dugme za rješavanje zagonetke ukoliko ona nije validna.

Kao i kod SolutionComponent komponente, tako i PuzzleInputComponent komponenta ima mogućnost prilagođavanja svoje veličine te će zauzeti maksimalan mogući prostor na mobilnim uređajima. Slika 3.22. prikazuje HTML predložak komponente koji se sastoji od dijela za upravljanje dimenzijama zagonetke i dijela za unos same zagonetke.

```
<div class="chooser">
  <label class="chooser__label"
    >Rows<input
      class="chooser__input"
      type="number"
      [value]="rows"
      (change)="changeRows($event)"
  /></label>
  <label class="chooser__label"
    >Columns<input
      class="chooser__input"
      type="number"
      [value]="columns"
      (change)="changeColumns($event)"
  /></label>
</div>

<div class="grid" [formGroup]="formGroup">
  <div
    formArrayName="matrix"
    *ngFor="
      let row of formMatrix.controls;
      let ri = index;
      trackBy: trackByIndex
    "
  >
    <div [formArrayName]="ri">
      <input
        *ngFor="
          let c of formRow(row).controls;
          let ci = index;
          trackBy: trackByIndex
        "
        [attr.aria-label]="'Input box:' + ri + ' ' + ci"
        [formControlName]="ci"
        type="text"
        class="grid__box"
        [style.fontSize.vw]="fontSize"
      />
    </div>
  </div>
</div>
```

Slika 3.22. HTML predložak PuzzleInputComponent komponente.

CameraComponent je jedina komponenta koju sadrži *CameraModule* modul. Glavna funkcija ove komponente je prepoznavanje zagonetke uz pomoć kamere. Kako bi to omogućila oslanja se na OpenCV.js i Tessereact.js biblioteke za obradu slike i detekciju znakova. Osim glavne funkcije,

ova komponenta omogućuje i mijenjanje kamere te odabir već spremljene slike iz datotečnog sustav operacijskog sustava.

Slika 3.23. prikazuje proces koji se odvija svaki put kada korisnik uslika sliku ili je odabere iz datotečnog sustava. Prvi korak procesa je označiti početak procesiranja zagonetke te tako dati korisniku povratnu informaciju o početku procesa. Proces čeka dok se OpenCV biblioteka ne učita i inicijalizira, te nakon toga sliku predaje algoritmu za prepoznavanje koji će pokušati napraviti ekstrakciju svih otoka iz zagonetke. Ako je zagonetka uspješno pronađena, onda će biti proslijeđena rješavaču zagonetke. Ukoliko je došlo do pogreške prilikom prepoznavanja zagonetke, korisniku će biti ispisana poruka s korakom na kojem je algoritam zastao.

```
private recognizeImage(imageAsDataUrl: string) {
  this.processing = true;
  this.error = false;
  this.notifications.next('Loading OpenCV');
  this.ngOpenCVService.isReady$
    .pipe(
      skipWhile(result => !result.ready),
      switchMap(() => {
        console.log('Loading image to canvas');
        return this.ngOpenCVService.loadImageToHTMLCanvas(
          imageAsDataUrl,
          this.canvasOutput.nativeElement
        );
      }),
      switchMap(() =>
        from(
          recognizePuzzle(this.canvasOutput.nativeElement, this.notifications)
        )
      ),
      tap(puzzle => {
        if (puzzle === false) {
          this.error = true;
          this.processing = false;
          this.cdRef.detectChanges();
        } else {
          console.log('Navigating to solver', puzzle);
          this.router.navigate([''], {
            queryParams: { puzzle: JSON.stringify(puzzle) },
            replaceUrl: true
          });
        }
      }),
      first()
    )
    .subscribe();
}
```

Slika 3.23. Rješavanje zagonetke iz predane slike.

WebcamComponent je komponenta koja omogućava jednostavno korištenje kamere na web platformi [17]. Glavne značajke ove komponente su prikaz videa iz kamere, snimanje fotografije, kompatibilnost s mobilnim uređajima na modernim operacijskim sustavima, pristup prednjim i

stražnjim kamerama (ukoliko postoji više kamera), zrcaljenje slike i mogućnost snimanja fotografije bez gubitka kvalitete.

4. TESTIRANJE I UPOTREBA WEB APLIKACIJE

U ovom poglavlju su prikazani rezultati testiranja web aplikacije te upute za njeno korištenje. Testiran je algoritam rješavanja zagonetke, prepoznavanje zagonetke iz slike i korisničko sučelje. Prilikom testiranja korisničkog sučelja obraćena je pozornost na brzinu učitavanja web aplikacije pri učitavanju na mobilnim uređajima kao i na računalima.

4.1. Testiranje algoritma rješavanja zagonetke

Testiranje algoritma za rješavanje zagonetke provedeno je na uzorku od 10 zagonetki koji se sastojao od zagonetki veličina 10x10 i 15x15, koje su sve različitih težina. Tablica 4.1. sadrži vremena rješavanja svake od zagonetke. Test je ponovljen tri puta kako bi se pronašla prosječna brzina rješavanja nad ovim uzorkom i ona iznosi 142,2 ms.

Tablica 4.1. Testiranje vremena rješavanje zagonetki

Redni broj	**Vrijeme izvršavanja u ms**		
	Test br. 1	Test br. 2	Test br. 3
1	76	91	63
2	87	129	93
3	125	114	101
4	143	106	142
5	88	91	90
6	163	160	154
7	154	114	144
8	118	97	106
9	241	266	207
10	352	239	212
Prosjek	**154.7**	**140.7**	**131.2**

Nadalje, potrebno je provjeriti točnost algoritma za rješavanje zagonetke. Test je proveden nad uzorkom od 50 zagonetki (Tablica 4.2.). Nakon testiranja pronađene su dvije zagonetke koji nisu mogle biti riješen, dok je ostalih 48 točno riješeno. Prema napravljenom testu algoritam ima uspješnost rješavanja zagonetki od 96%. Problem prilikom rješavanja predstavljaju zagonetke koje u jednom trenutno nemaju sigurnu mogućnost za spajanje dva otoka. Kako bi se riješile te zagonetke potrebno je povući nasumično jedan most te pokušati naći rješenje zagonetke. Ukoliko potpuno rješenje nije pronađeno, zagonetka se mora vratiti u stanje prije nasumičnog povlačenja mosta i pokušati s nekim drugim nasumičnim mostom. Implementirani algoritam ne sadrži logiku za predviđanje mogućih mostova te iz tog razloga i ne može riješiti zagonetke tog tipa.

Tablica 4.2. Testiranje točnosti rješavanja zagonetke

Redni broj	**Težina i veličina zagonetke**				
	10x10 normalno	10x10 teško	15x15 lagano	15x15 normalno	15x15 teško
1	DA	DA	DA	DA	DA
2	NE	DA	DA	DA	DA
3	DA	DA	DA	DA	DA
4	DA	DA	DA	DA	DA
5	DA	DA	DA	DA	DA
6	DA	DA	DA	DA	DA
7	DA	DA	DA	DA	DA
8	DA	DA	DA	NE	DA
9	DA	DA	DA	DA	DA
10	DA	DA	DA	DA	DA
Prosjek	**90%**	**100%**	**100%**	**90%**	**100%**

4.2. Testiranje prepoznavanja zagonetke preko slike

Testiranje prepoznavanje zagonetke preko slike koja je uslikana s mobilnim uređajem unutar web aplikacije provedeno je nad uzorkom od 10 zagonetki (5 zagonetki veličine 10x10 i 5 zagonetki veličine 15x15). Tablica 4.3. sadrži vremena i broj pokušaja iz kojeg je zagonetka uspješno

prepoznata. Uspoređivanjem vremena prvih pet s drugih pet riješenih zagonetki primjetan je porast vremena prepoznavanja za 200 ms. Osim toga, prilikom testiranja primijećena je potreba za slikanjem zagonetke više puta. Iako algoritam za prepoznavanje sadrži mogućnost za ispravljanje iskrivljenja slike, taj algoritam ima i svoje nedostatke. Zbog iskrivljenja slike, lošeg osvjetljenja i/ili lošeg fokusiranja slike može doći do prepoznavanja pogrešnih krugova te neprepoznavanja krugova. Testiranjem je i utvrđena nemogućnost prepoznavanja otoka pri velikim veličinama zagonetke (testirano na 25x25). Taj nedostatak proizlazi iz male rezolucije slike koja se dobiva tijekom slikanja kroz web aplikaciju. Trenutno na web platformi ne postoji mogućnost za dobivanje izvorne slike tijekom slikanja, već se slika dobiva tako što se zatraži video od kamere i iz kojega se onda izvlači jedna slika. Video koji ima *Full HD* rezoluciju (1920x1080 piksela) sadrži značajno manje piksele od slike koja je izvorno uslikana (najčešće preko 8 milijuna piksela). Zbog manje rezolucije, slika ima manje detalja te je teže prepoznati krugove, ali i brojeve koji se nalaze unutar krugova.

Tablica 4.3. Vrijeme potrebno za prepoznavanje zagonetke iz slike.

Redni broj	Vrijeme prepoznavanja u ms	Broj pokušaja
1	1944	1
2	2087	1
3	1989	1
4	1907	2
5	1995	1
6	2118	1
7	2193	2
8	2063	1
9	2210	3
10	2191	1
Prosjek	**2069.7**	**1.4**

4.3. Testiranje korisničkog sučelja

4.3.1. Brzina učitavanja web aplikacije

Za testiranje brzine učitavanje web aplikacije korišteni su razvojni alati koji se nalaze unutar Google Chrome Internet preglednika. Razvojni alati omogućuju niz funkcionalnosti poput mijenjanja strukture stranice, pregleda učitanih resursa, pregleda iskorištene memorije, prepoznavanje grešaka, simulacije stranice na drugim uređajima i sl. Jedna od tih funkcionalnosti je i provjera brzine učitavanja web aplikacije. Nakon izvršene provjere, alat kao rezultat vraća vremena ključnih događaja tijekom učitavanja stranice [18]. Ti događaji su:

- FP – (engl. *First Paint*) vrijeme kada se tijekom učitavanja web stranice prvi put nešto prikaže na zaslonu tj. kada se na zaslonu prikaže barem jedan piksel (npr. to može biti pozadinska boja web stranice)
- DCL – (engl. *DOM Content Loaded*) – ovaj događaj se aktivira kada se inicijalni HTML dokument u potpunosti učita i raščlani (engl. *parse*) bez čekanja na učitavanje slika, stilskih dokumenata i poddokumenata [19]
- FCP – (engl. *First Contentful Paint*) ovaj događaj se aktivira onog trenutka kada Internet preglednik prvi put prikaže bilo koji dio iz DOM-a koji može biti tekst, slika ili neki drugi element
- L – (engl. *Load*) ovaj događaj se aktivira kada je učitan sav sadržaj web stranice, uključujući sve zavisne resurse kao što su skripte, slike i stilski dokumenti [20]

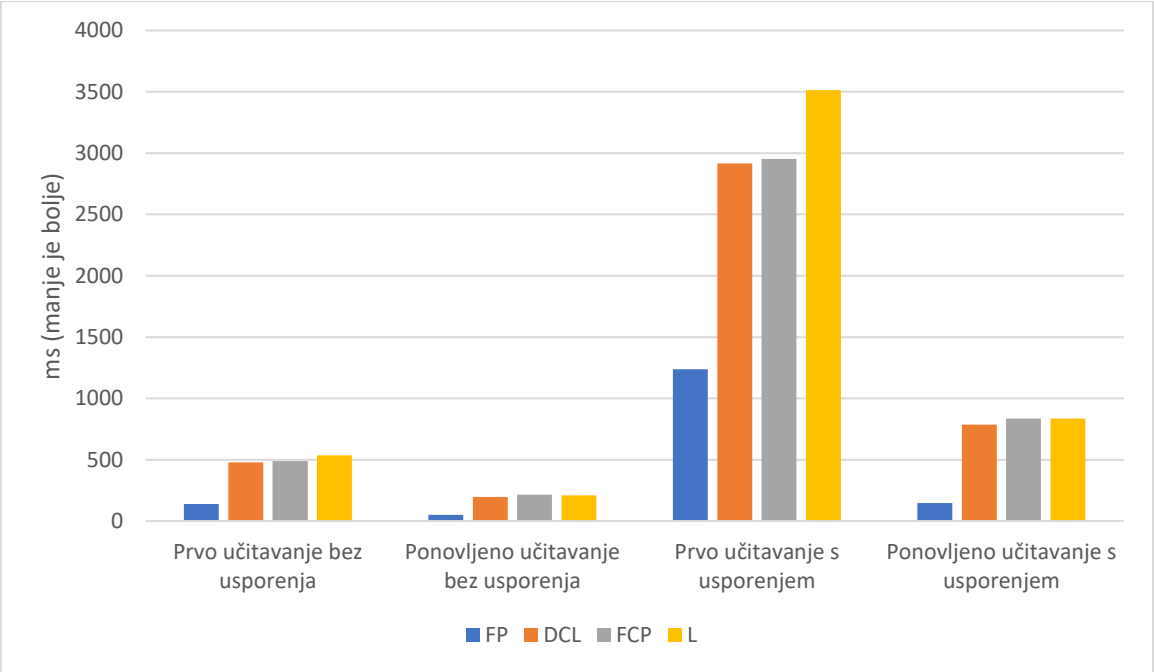
Tablica 4.4. sadrži testiranje učitavanja brzine koje je provedeno u Internet pregledniku Google Chrome verzije 76 s Microsoft Windows 10 operacijskim sustavom. Test je pokretan na prijenosnom računalu s Intel Core i5-7200 CPU-om s frekvencijom od 2,5 GHz te 8 GB radne memorije. Prilikom testiranja korištena je DSL internetska veza s brzinom preuzimanja od 8 Mbit/s. Kako bi se pronašla i brzina učitavanja web aplikacije na mobilnim uređajima napravljeno je i testiranje s usporenjem koje se sastojalo od usporenja CPU-a i brzine interneta za faktor od četiri.

Tablica 4.4. Vremena ključnih događaj prilikom učitavanja web aplikacije.

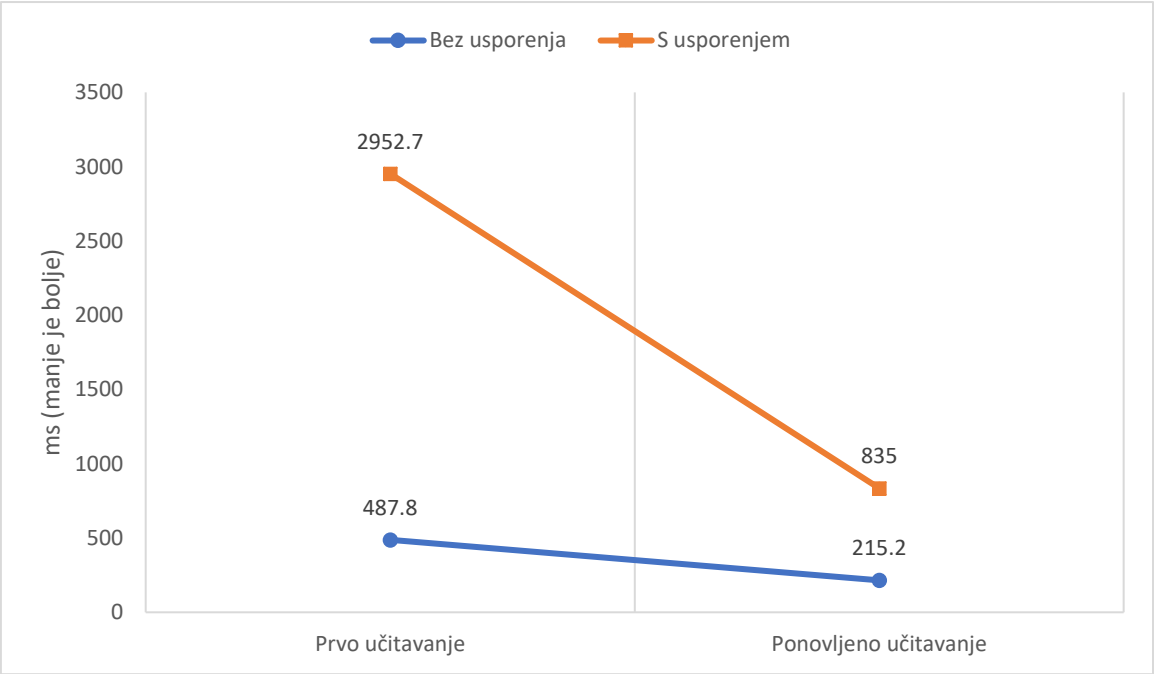
	FP	DCL	FCP	L
Prvo učitavanje bez usporenja	139,4	478	487,8	534,7
Ponovljeno učitavanje bez usporenja	49,2	197,4	215,2	210,5
Prvo učitavanje s usporenjem	1236,9	2915,9	2952,7	3512,8
Ponovljeno učitavanje s usporenjem	146,3	786,6	835	836

Tijekom testiranja testirano je prvo učitavanje web aplikacije (kada Internet preglednik nema spremljen niti jedan resurs od web aplikacije u međuspremniku) i ponovljeno učitavanje (kada Internet preglednik ima spremljene resurse u međuspremniku). Slika 4.1. grafički prikazuje vremena učitavanja web aplikacije te ukazuje na veliko poboljšanje brzine učitavanja prilikom ponovljenog učitavanja. FCP je smanjen s 487,8 na samo 215,2 ms (smanjenje za 56 %), dok na simuliranom mobilnom uređaju to smanjenje je još veće. Naime, smanjeno je s 2952,7 na 835 ms što u postotcima iznosi 72 % (Slika 4.2.).

Tako dobri rezultati prilikom ponovljenog učitavanja web aplikacije ostvareni su agresivnim spremanjem aplikacijskih resursa u međuspremnik Internet preglednika. Uz pomoć Service Workera spremljeni su svi resursi: inicijalni HTML dokument, JavaScript i CSS dokumenti te sve slike koje se prikazuju na stranici. Kada korisnik pokuša pristupiti web aplikaciji po drugi put ona je u potpunosti spremljena te se u jako kratkom roku učita - ispod jedne sekunde. Internet preglednik uopće ne šalje zahtjev poslužitelju jer sve potrebne informacije o web aplikaciji već postoje.



Slika 4.1. Graf ključnih trenutaka za prvo i ponovljeno učitavanje web aplikacije.

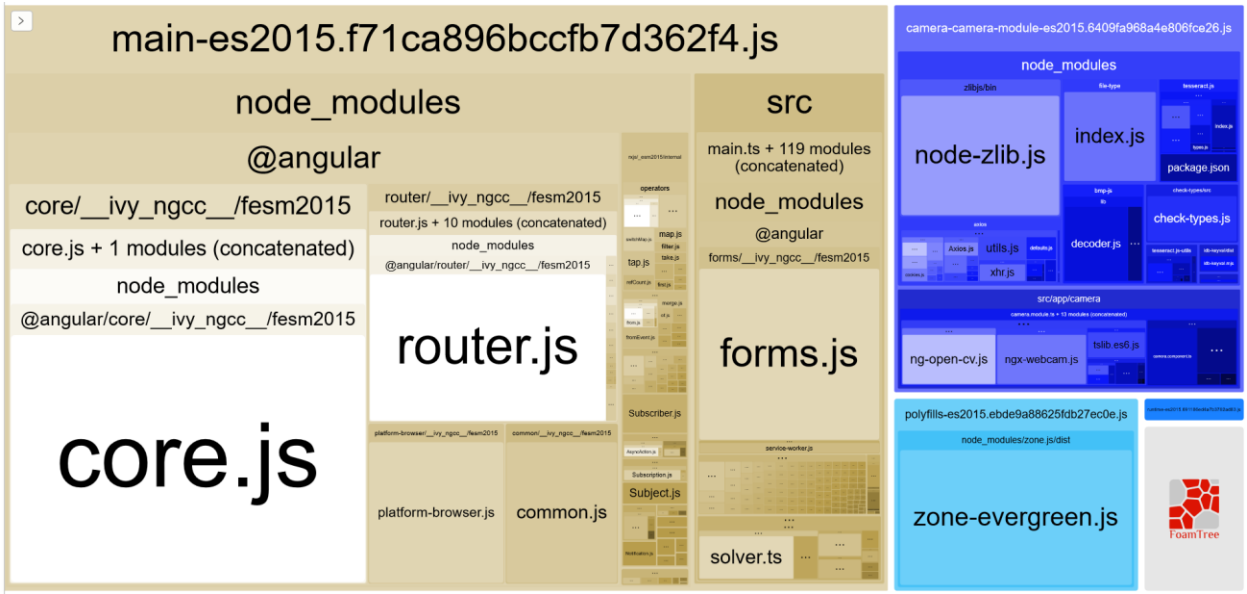


Slika 4.2. Graf poboljšanja vremena učitavanja web aplikacije prilikom ponovljenog učitavanja.

4.3.2. Pregled datoteka web aplikacije

Procesom izgradnje Angular aplikacije stvara se nekoliko JavaScript datoteka, tzv „komadi“ (engl. *chunks*). Time se izbjegava problem zahtijevanja prevelikog broja datoteka prilikom učitavanja aplikacije. Izgradnjom se sve JavaScript i TypeScript datoteke spajaju u jednu cjelinu i sažimaju. Rezultat tog procesa su sljedeće JavaScript datoteke (Slika 4.3.):

- runtime.js – Webpack [21] datoteka koja omogućuje dohvaćanje komada
- polyfills.js – datoteka s implementacijama Web API-ja koji nisu podržani unutar svih Internet preglednika
- main.js – glavna datoteka unutar koje se nalazi implementacija primarnog dijela aplikacije (Angular programski okvir, komponente, rješavač zagonetke…)
- camera.module.js – datoteka unutar koje se nalazi implementacija zaslona za slikanje zagonetke (sadrži komponentu kamere, OpenCV i Tesseract servise)



Slika 4.3. Stvorene JavaScript datoteke nakon izgradnje Angular aplikacije s njihovim zavisnim modulima (veličina je proporcionalna količini bajtova koji pojedini komad zauzima).

Osim tih JavaScript datoteka, stvorene su i sporedne datoteke ngsw-worker.js, ngsw.json, safety-woker.js i worker-basic.min.js koje su zaslužne za pokretanje Service Workera te instalaciju web aplikacije. Također, tijekom izgradnje se stvara i styles.css datoteka koja sadrži informacije o prikazu aplikacije.

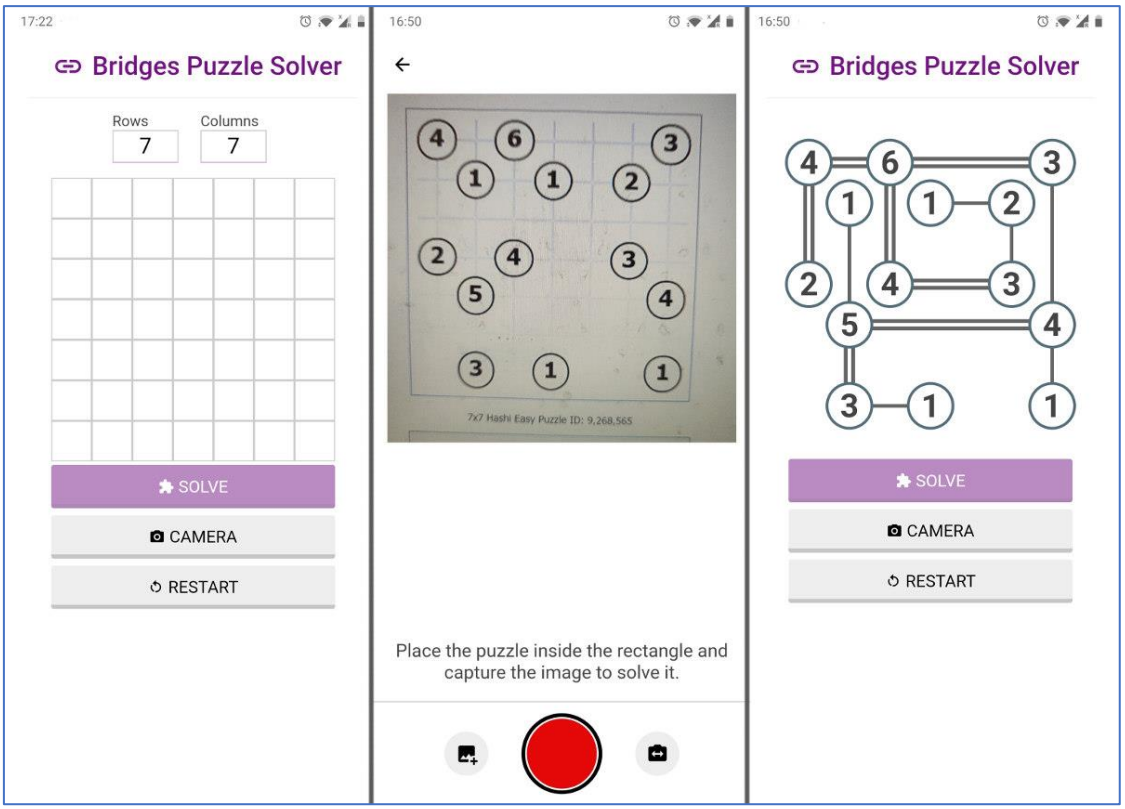
Tijekom prepoznavanja zagonetke iz slike učitavaju se biblioteke OpenCV.js i Tesseract.js s njihovim popratnim WebAssembly datotekama. Veličina OpenCv.js datoteke iznosi 534 kB, dok *opencv_js.wasm* datoteka iznosi 5,3 MB jer sadrži cijelu implementaciju OpenCV biblioteke koja je kompilirana u WebAssembly. S druge strane, za uspostavljanje Tesseract OCR *enginea* potrebno je učitati WebAssembly datoteku veličine 3,1 MB i prilagođeni skup podataka za prepoznavanje engleskog jezika veličine 10,4 MB.

Među preostalim datotekama su ikone različitih veličina koje predstavljaju logo aplikacije, SVG ikone koje se koriste unutar gumbova, MP3 datoteka sa zvukom klika (zvuk koji se čuje kada se uslika zagonetka) te webmanifest datoteka koja sadrži informacije o web aplikaciji.

4.3.3. Prilagodljivost različitim veličinama zaslona

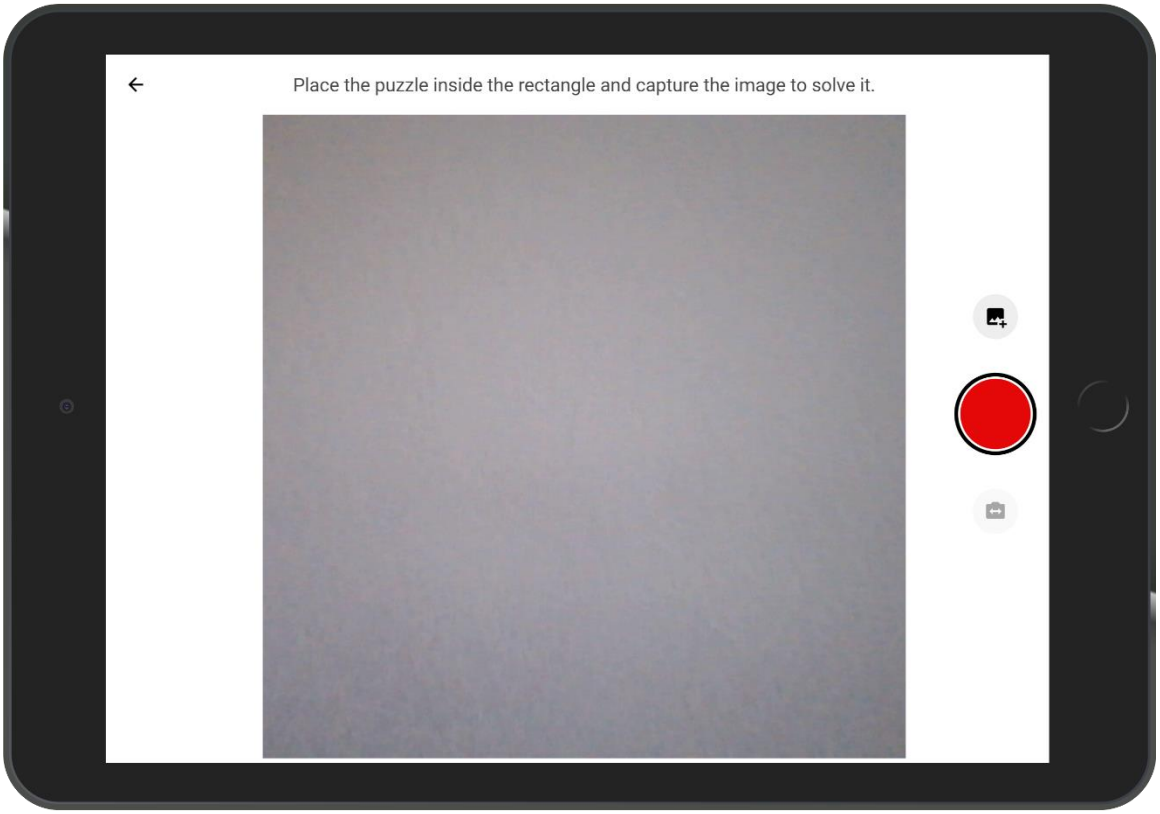
Korisnici web aplikaciju mogu koristiti unutar bilo kojeg Internet preglednika i bilo kojeg uređaja. Najčešći uređaji na kojima se koristi aplikacija su stolno računalo, prijenosna računala, tableti i pametni telefoni. Zbog funkcionalnosti slikanja zagonetke kamerom kroz web aplikaciju odabran je pametni telefon kao primarni uređaj prema kome je prvenstveno dizajnirana aplikacija. Izabran je kao primarni uređaj zbog jednostavne mogućnosti slikanja koju ostali uređaji ne nude.

Slika 4.4. prikazuje aplikaciju u tri stanja u kojima se može nalaziti: početno stanje s praznim poljima za unos, stanje slikanja zagonetke i stanje rješenja zagonetke. Za stanje aplikacije u načinu slikanja, gumbovi su prilagođeni mobilnim uređajima kako bi korisnik mogao što jednostavnije uslikati zagonetku.



Slika 4.4. Tri prikaza aplikacije na mobilnim uređajima (lijevo – početni zaslon, sredina – zaslon za slikanje zagonetke, desno – zaslon s rješenjem zagonetke).

Sljedeći uređaj koji je veći od pametnog telefona je tablet. Izabrana je prijelazna točka širine od 768 piksela nakon koje se primjenjuje izgled aplikacije za veće zaslone. Ono što je promijenjeno jest pozicija dugmadi za kontrolu kamere, koja prelazi s dna na desnu stranu zaslona. Svi uređaji koji imaju širinu zaslona veću od 768 piksela (tableti, prijenosna računala, stolna računala i sl.) imat će promijenjen izgled u odnosu na pametne telefone kada je aplikacija u stanju slikanja zagonetke. Slika 4.5. prikazuje aplikacijsko stanje slikanja zagonetke kamerom na tablet uređaju.



Slika 4.5. Prikaz aplikacije u stanju slikanja zagonetke na tablet uređaju.

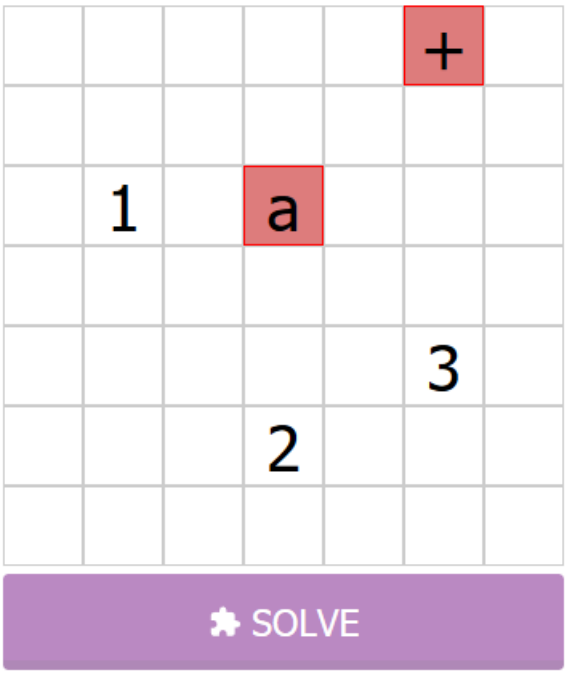
4.4. Upute za korištenje

4.4.1. Ručni unos podataka

Kako bi se uspješno riješila zagonetka tako što se ručno unesu podatci, potrebno je slijediti sljedeće korake:

1. Unos dimenzija zagonetke u polja s oznakama *Rows* i *Columns*
2. Unos brojeva od 1 do 8 za svaki otok unutar polja za unos otoka (dva otoka ne mogu postojati direktno jedan pored drugoga jer onda ih nije moguće spojiti)
3. Pritisak na gumb s nazivom *Solve*

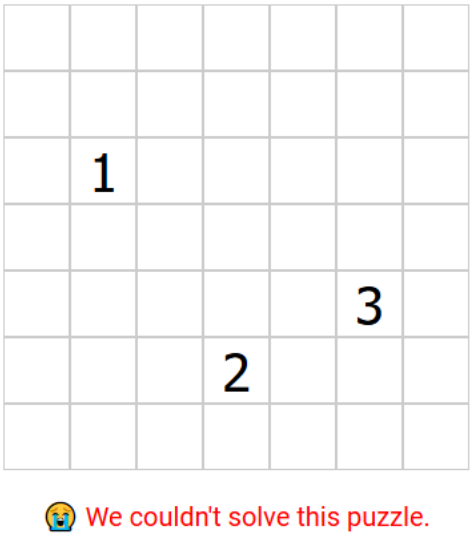
Ukoliko se u neko od polja za unos brojeva otoka unese neki nedozvoljeni znak poput slova, to polje dobiva crvenu pozadinu kako bi se korisnika obavijestilo da je uneseni znak nedozvoljen. Osim toga, dugme za rješavanje postaje onemogućeno te mijenja svoju boju kako ga korisnik ne bi mogao više pritisnuti (Slika 4.6.).



Slika 4.6. Stanje polja za unos otoka tijekom s prisustvom nedozvoljenih znakova.

Dugme s nazivom *Restart* briše sve što je uneseno u polje unosa te omogućuje unos nove zagonetke, dok dugme s nazivom *Camera* korisnika vodi na zaslon za prepoznavanje zagonetke preko kamere.

Ukoliko korisnik unese zagonetku koju rješavač ne može riješiti, ispod polja za unos će biti ispisana poruka o nemogućnosti rješenja zagonetke (Slika 4.7.).



Slika 4.7. Obavijest korisniku o nemogućnosti pronalaska rješenja zagonetke.

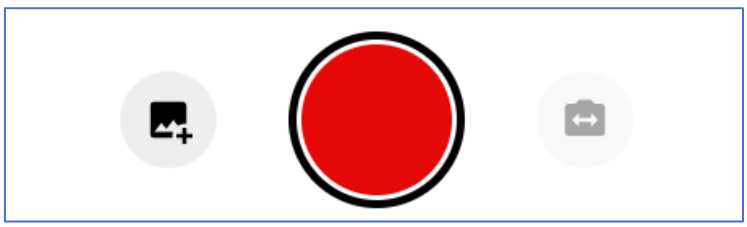
4.4.2. Automatski unos podataka iz slike

Prilikom automatskog unosa zagonetke uz pomoć kamere korisnik sve što mora učiniti je postaviti zagonetku unutar određenog kvadrata te stisnuti dugme za slikanje. Ukoliko je obrada slike uspješno završena te brojevi otoka prepoznati aplikacija korisnika vraća na početni zaslon gdje je prikazano rješenje zagonetke. Prvi put kada korisnik bude pristupao zaslonu s kamerom bit će upitan želi li dopustiti aplikaciji pristup kameri te na to pitanje mora odgovoriti potvrdno. Ako korisnik odbije pristup, onda neće moći slikati zagonetku kamerom.

U slučaju da se na jednom od koraka pri prepoznavanju zagonetke dogodi greška, ta greška će biti ispisana crvenim slovima na zaslonu. Korisnik u tom slučaju može pokušati ponovo uslikati zagonetku tako što će bolje pozicionirati zagonetku unutar predviđenog kvadrata. Ako je greška pri dohvaćanju OpenCV ili Tesseract biblioteke, korisnik treba provjeriti svoju Internet konekciju.

Sučelje zaslona za prepoznavanje uz pomoć kamere sastoji se od tri glavna gumba (Slika 4.8.):

- Crveni gumb u sredini omogućuje korisniku slikanje zagonetke
- Gumb za odabir slikovne datoteke
- Gumb za mijenjanje kamere (ukoliko korisnik ima samo jednu kameru ovaj gumb je onemogućen)



Slika 4.8. Akcijski gumbi na zaslonu za slikanje zagonetke uz pomoć kamere.

Osim ova tri glavna gumba, u gornjem lijevom kutu nalazi se i gumb za vraćanje aplikacije na početni zaslon.

5. ZAKLJUČAK

U ovom diplomskom radu obrađena je implementacija web aplikacije za rješavanje logičke zagonetke *Hashiwokakero* (mostovi). Cilj diplomskog rada bio je izrada web aplikacije koja će se ponašati kao *native* aplikacija na mobilnim uređajima te omogućiti napredne funkcionalnosti koje prije par godina nisu bile dostupne na web platformi.

Implementirana je aplikacija s jednostavnim korisničkim sučeljem kroz koje se može ručno unijeti zagonetka ili snimiti slika uz pomoć kamere te nakon toga automatski riješiti. Prikazan je način na koji se napredne biblioteke poput *OpenCV*-a i *Tesseracta* mogu pokrenuti na web platformi i omogućiti naprednu obradu slike i prepoznavanje znakova (brojeva) iz slike. Pametnim spremanjem svih resursa web aplikacije i korištenjem WebAssembly ostvarena je aplikacija visokih performansi. Testiranjem je utvrđena brzina učitavanje aplikacije te potrebna funkcionalnost koja je zadana ovim radom.

LITERATURA

[1] Anonimno, »Conceptis Puzzles,« [Mrežno]. Dostupno na: https://www.conceptispuzzles.com/index.aspx?uri=puzzle/hashi/history. [Pokušaj pristupa 10. lipanj 2019.].

[2] Anonimno, »WEB Nicoli,« NIKOLI Co., [Mrežno]. Dostupno na: http://www.nikoli.co.jp/en/puzzles/hashiwokakero.html. [Pokušaj pristupa 10. lipanj 2019.].

[3] Anonimno, »Wikipedia,« [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Hashiwokakero. [Pokušaj pristupa 10. lipanj 2019.].

[4] Anonimno, »HTML: Hypertext Markup Language,« Mozilla, [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTML. [Pokušaj pristupa 10. lipanj 2019.].

[5] Anonimno, »About JavaScript,« Mozilla, [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Pokušaj pristupa 10. lipanj 2019.].

[6] Anonimno, »Wikipedia,« [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/TypeScript. [Pokušaj pristupa 15. kolovoz 2019.].

[7] C. B. E. T. B. Z. Ghao, »To type or not to type: quantifying detectable bugs in JavaScript,« u *ICSE '17 Proceeding of the 39th International Conferance on Software Engineering*, Buenos Aures, 2017..

[8] Anonimno, »CSS: Cascading Style Sheets,« Mozilla, [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/CSS. [Pokušaj pristupa 10. lipanj 2019.].

[9] Anonimno, »WebAssembly,« Mozilla, [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/WebAssembly. [Pokušaj pristupa 11. lipanj 2019.].

[10] D. Gavigan, »Medium,« [Mrežno]. Dostupno na: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7. [Pokušaj pristupa 15. kolovoz 2019.].

[11] I. Bodrov-Krukowski, »Sitepoint,« [Mrežno]. Dostupno na: https://www.sitepoint.com/angular-introduction/. [Pokušaj pristupa 15. kolovoz 2019.].

[12] Anonimno, »OpenCV,« OpenCV team, [Mrežno]. Dostupno na: https://opencv.org/about/. [Pokušaj pristupa 11. lipanj 2019.].

[13] Anonimno, »OpenCV: Introduction to OpenCV.js and Tutorials,« OpenCV team, [Mrežno]. Dostupno na: https://docs.opencv.org/3.4/df/d0a/tutorial_js_intro.html. [Pokušaj pristupa 11. lipanj 2019.].

[14] Anonimno, »Emscripten,« [Mrežno]. Dostupno na: https://emscripten.org/. [Pokušaj pristupa 11. lipanj 2019.].

[15] Anonimno, »TESSERACT.js,« [Mrežno]. Dostupno na: https://tesseract.projectnaptha.com/. [Pokušaj pristupa 11. lipanj 2019.].

[16] Anonimno, »Tesseract (software),« Wikipedia, [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Tesseract_(software). [Pokušaj pristupa 11. lipanj 2019.].

[17] Anonimno, »Github,« [Mrežno]. Dostupno na: https://github.com/basst314/ngx-webcam. [Pokušaj pristupa 10. kolovoz 2019.].

[18] Philip Walton, »Google Developers,« Google, 29 svibanj 2019. [Mrežno]. Dostupno na: https://developers.google.com/web/fundamentals/performance/user-centric-performance-metrics#first_paint_and_first_contentful_paint. [Pokušaj pristupa 5. kolovoz 2019.].

[19] Anonimno, »Document: DOMContentLoaded event,« Mozilla, [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event. [Pokušaj pristupa 5 kolovoz 2019].

[20] Anonimno, »Window: load event,« Mozilla, [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event. [Pokušaj pristupa 5 kolovoz 2019].

[21] Anonimno, »webpack,« [Mrežno]. Dostupno na: https://webpack.js.org/concepts/. [Pokušaj pristupa 5. kolovoz 2019.].

POPIS TABLICA I SLIKA

Tablice

Tablica 3.1. Mapiranje znakova rješenja zagonetke u polja zagonetku. 27
Tablica 4.1. Testiranje vremena rješavanje zagonetki 31
Tablica 4.2. Testiranje točnosti rješavanja zagonetke 32
Tablica 4.3. Vrijeme potrebno za prepoznavanje zagonetke iz slike. 33
Tablica 4.4. Vremena ključnih događaj prilikom učitavanja web aplikacije. 35

Slike

Slika 2.1. Rješenje zagonetke mostovi veličine 7x7 (hashiwokakero). 2
Slika 3.1. Primjer Angular komponente koja predstavlja listu stvari koje treba obaviti. 7
Slika 3.2. Primjer ulazne dvodimenzionalne matrice algoritma za rješavanje zagonetke. 9
Slika 3.3. Primjer izlazne vrijednosti rješavača zagonetke. 10
Slika 3.4. Struktura grafa korištena unutar algoritma za rješavanje zagonetke. 11
Slika 3.5. TraverseUp funkcija za pronalazak mostova i otoka iz matrice. 12
Slika 3.6. Provjera riješenosti zagonetke. 13
Slika 3.7. Postavljanje pomoćnih varijabli. 13
Slika 3.8. Svi uvjeti za koje se povlače mostovi između otoka. 14
Slika 3.9. Zagonetka nad kojom će se vršiti primjer algoritma detekcije zagonetke iz slike. 16
Slika 3.10. Uslikana zagonetka pomoću mobilnog uređaja. 17
Slika 3.11. Detektirani krugovi unutar zagonetke metodom Houghove transformacije (detektirani krugovi su označeni crvenom bojom). 18
Slika 3.12 Slika zagonetke nakon što je uklonjeno iskrivljenje i u kojoj su crvenom bojom označeni detektirani krugovi. 19
Slika 3.13. Funkcija koja vraća dimenzije zagonetke. 20
Slika 3.14. Kod izrezivanja broja iz slike. 21
Slika 3.15. Prepoznati brojevi iz zagonetke koji su spojeni u jednu cjelinu. 21
Slika 3.16. Prepoznavanje brojeva iz slike. 22
Slika 3.17. Početni modul Angular aplikacije. 23
Slika 3.18. Hijerarhijsko stablo komponenti i modula aplikacije 24
Slika 3.19 Definicija putanja unutar aplikacije. 25
Slika 3.20. HTML predložak HomeComponent komponente. 26

Slika 3.21. HTML predložak SolutionComponent komponente. 27
Slika 3.22. HTML predložak PuzzleInputComponent komponente. 28
Slika 3.23. Rješavanje zagonetke iz predane slike. 29
Slika 4.1. Graf ključnih trenutaka za prvo i ponovljeno učitavanje web aplikacije. 36
Slika 4.2. Graf poboljšanja vremena učitavanja web aplikacije prilikom ponovljenog učitavanja. 36
Slika 4.3. Stvorene JavaScript datoteke nakon izgradnje Angular aplikacije s njihovim zavisnim modulima (veličina je proprcionalna količini bajtova koji pojedini komad zauzima). 37
Slika 4.4. Tri prikaza aplikacije na mobilnim uređajima (lijevo – početni zaslon, sredina – zaslon za slikanje zagonetke, desno – zaslon s rješenjem zagonetke). 39
Slika 4.5. Prikaz aplikacije u stanju slikanja zagonetke na tablet uređaju. 40
Slika 4.6. Stanje polja za unos otoka tijekom s prisustvom nedozvoljenih znakova. 41
Slika 4.7. Obavijest korisniku o nemogućnosti pronalaska rješenja zagonetke. 41
Slika 4.8. Akcijski gumbi na zaslonu za slikanje zagonetke uz pomoć kamere. 42

SAŽETAK

Tema ovog diplomskog rada je web aplikacija za rješavanje logičke zagonetke Hashiwokakero. Cilj rada je implementacija jednostavnog korisničkog sučelja koje će maksimalno napredne mogućnosti web platforme. Aplikacija omogućuje ručni i automatskih unos zagonetke – učitavanje snimanjem slike uz pomoć kamere. Pri detekciji zagonetke iz slike koristi se OpenCV.js za obradu slike i Tesseract.js za prepoznavanje znakova iz slike. Angular programski okvir pokreće aplikaciju koja je prvenstveno napravljena za mobilne uređaje, ali podržava i sve ostale. Testiranjem aplikacije utvrđena je brzina aplikacije koja je u skladu s izvornim aplikacija.

Ključne riječi: Hashiwokakero, OpenCV, Tesseract, Angular, WebAssembly, rješavač

ABSTRACT

WEB APPLICATION FOR SOLVING THE LOGIC PUZZLE "HASHIWOKAKERO"

The subject of this final paper is a web application for solving the logic puzzle Hashiwokakero. The goal of this paper was the implementation of a simple user interface that utilizes many advanced technologies available on the web platform. The application allows manual and automatic puzzle input – loading the image by capturing a photo with the camera. OpenCV.js is used for image manipulation and Tesseract.js for character recognition when detecting the puzzle from a photo. The Angular framework is used to power the implemented mobile-first application. The test results show that the application is as performant as other native applications.

Keywords: Hashiwokakero, OpenCV, Tesseract, Angular, WebAssembly, solver

ŽIVOTOPIS

Antonio Vrbić rođen je 8. lipnja 1995. godine u Žepču. Pohađao je osnovnu školu u Žepču u razdoblju od 2002. do 2010. godine. Nakon osnovne škole pohađa Katolički školski centar „Don Bosco“ Žepče, smjer Tehničar za mehatroniku, u razdoblju od 2010. do 2014. godine. Nakon završene srednje škole, 2014. godine upisuje sveučilišni preddiplomski studij Računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Preddiplomski studij završava 2017. godine i stječe zvanje prvostupnika inženjera računarstva. Iste te godine upisuje diplomski studij Računarstva smjer Podatkovne i informacijske znanosti na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek.

PRILOZI (na CD-u)

Prilog 1. Datoteke pisanog dijela rada (docx i pdf)

Prilog 2. Programski kod