UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Darko Lipovec

Vizualizacija in animacija preiskovalnega algoritma RBFS

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. dr. Marko Robnik Šikonja

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00047/2010
Datum: 05.11.2010

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

Kandidat: **DARKO LIPOVEC**

Naslov: **VIZUALIZACIJA IN ANIMACIJA PREISKOVALNEGA ALGORITMA RBFS**

VISUALIZATION AND ANIMATION OF RBFS SEARCH

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Pri poučevanju preiskovalnih algoritmov potrebujemo orodja za boljšo predstavitev vsebin in za njihovo lažje razumevanje. Algoritem RBFS ima enaka zagotovila glede optimalnosti kot algoritem A*, a porabi bistveno manj pomnilnika. Namesto hranjenja stanj uporablja vračanje ter posodabljanje zgornje meje iskanja. Njegovo delovanje je mnogo lažje razumeti z vizualizacijo na izbranih primerih.
Proučite in intuitivno opišite delovanje algoritma RBFS ter njegovo delovanje ilustrirajte na nekaj poučnih problemih, ki omogočajo vizualno prepričljivo predstavitev. Izdelajte animacijo iskanja optimalne rešitve, ki prikazuje razvoj vozlišč preiskovalnega drevesa in izvajanje psevdokode algoritma. Razvite primere vključite v spletno interaktivno aplikacijo, ki uporabniku omogoča nastavitev začetnega stanja in konfiguracijo prostora stanj.

Mentor:

prof. dr. Marko Robnik Šikonja

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Darko Lipovec,
z vpisno številko 63040436,

sem avtor diplomskega dela z naslovom:
Vizualizacija in animacija preiskovalnega algoritma RBFS

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  prof. dr. Marka Robnik Šikonje

- 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 8. 2. 2011 Podpis avtorja: _____

Zahvala

Zahvaljujem se mentorju prof. dr. Marku Robnik Šikonji za strokovno usmerjanje in pomoč pri izdelavi naloge.

Hvala tudi vsem najbližjim, ki so mi stali ob strani, hkrati se jim opravičujem za mojo nedružabnost v času pisanja diplomske naloge.

Kazalo

Povzetek

Cilj diplomskega dela je izdelava spletne aplikacije, ki služi kot pripomoček za boljše razumevanje delovanja algoritma RBFS. Najprej predstavimo lastnosti in delovanje algoritma RBFS in opišemo Adobe Flash razvojno platformo, ki smo jo uporabili za razvoj aplikacije. Predstavimo orodje za animacijo Flash in razvojno orodje Flash Builder. Opišemo grafični uporabniški vmesnik in tri animirane primere preiskovanja algoritma RBFS. Predstavimo obnašanje algoritma z animacijo izvajanja psevdokode ter reševanja problema iskanja najkrajše poti med dvema krajema. Omogočen je tudi vpogled v dogajanje v pomnilniku, kjer algoritem RBFS med preiskovanjem gradi drevo. V zaključku povzamemo ugotovitve in predstavimo ideje za nadaljnje izboljšave spletne aplikacije in animacij algoritma RBFS.

Ključne besede:
algoritem RBFS, animacija, vizualizacija, hevristično preiskovanje

Abstract

We created a web application for better comprehension of the RBFS algorithm. We present RBFS algorithm and its properties and describe the Adobe Flash development platform, which we used for the development together with the tools for Flash animation and Flash Builder development environment. We present a graphical user interface and three animated examples where RBFS searches for an optimal solution. The behavior of the algorithm is illustrated using an animation of the pseudo-code and animation for finding the shortest path between two points. Inner workings of the algorithm show the data structures used by the algorithm during the search. In the closing chapter we overview the findings and present the ideas for further improvements of the RBFS algorithm animations.

Keywords:
RBFS algorithm, animation, visualization, heuristic search

1. Uvod

Vzroki za težavnost učenja so obsežnost in zapletenost vsebin ter nepoznavanje področja. Učna snov je največkrat v obliki tekstovnega gradiva. Tekstovno predstavljena vsebina je lahko natančna, vendar je težje razumljiva, saj zahteva dobro koncentracijo in prilagojenost takšnemu podajanju snovi. Vizualizacije učnih gradiv v obliki grafov, tabel in drugih slikovnih gradiv poleg lažjega razumevanja kompleksnejših vsebin, omogočajo tudi lažji priklic informacij, zlasti vizualnim tipom učencev. Tudi snov o algoritmih in podatkovnih strukturah je največkrat predstavljena s tekstovnimi vsebinami. Le redko vizualizacije te snovi vključujejo animacije, simulacije in video vsebine tako, da lahko opazimo delovanje algoritma. Pomanjkljivost obstoječih vizualizacij je nezadostna dinamičnost, omejenost opazovanja algoritma le na prostorsko in časovno zahtevnost ali nezmožnost aktivnega sodelovanja.

Želja po izboljšanju gradiv, njihovi vizualizaciji in posledično lažjem razumevanju je motivacija za to diplomsko delo. Ukvarjamo se z vizualizacijo *algoritma rekurzivnega iskanja po načelu „najprej najboljši“* (ang. Recursive Best-First Search – RBFS), ki je predstavljen s samostojno spletno aplikacijo.

V prvem poglavju diplomske naloge se poučimo o algoritmu RBFS. Seznanimo se z njegovimi značilnostmi, s hevristično informacijo, s katero si RBFS pomaga pri preiskovanju in s prostorom stanj, s katerim je problem predstavljen. Za lažjo predstavo opišemo delovanje algoritma na problemu igre drsečih ploščic.

V drugem poglavju predstavimo razvojno platformo Adobe Flash, ki je namenjena predvsem razvoju animacij, spletnim stranem in drugim zabavnim vsebinam. Predstavimo razvojna orodja Flash, Flash Builder in predvajalnik izvoznih datotek Flash Player. Opišemo nekaj primerov uporabe te tehnologije v praksi ter omenimo nekaj alternativnih pristopov.

V tretjem poglavju predstavimo našo aplikacijo. Z delovanjem RBFS se seznanjamo skozi tri različne primere:

1. RBFS spremljamo preko izvajanja stavkov v psevdokodi in vzporedno opazujemo stanje v pomnilniku,
2. prikazujemo, kako pri reševanju danega problema RBFS gradi drevo,
3. RBFS preiskusimo na delu zemljevida Ljubljane.

Tako dobi uporabnik širšo predstavo o delovanju algoritma RBFS, kar je glavni cilj diplomske naloge.

V zadnjem poglavju predstavimo sklepne ugotovitve ter možne dopolnitve in izboljšave spletne aplikacije.

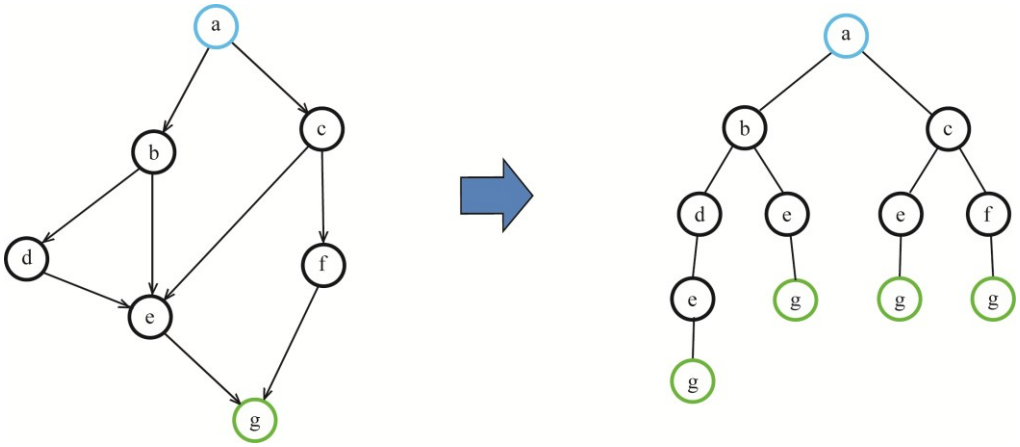
2. Iskalni algoritem RBFS

2.1. Osnovni pojmi RBFS algoritma

Algoritem *rekurzivnega iskanja po načelu „najprej najboljši“* (ang. Recursive Best-First Search – RBFS) je iskalni algoritem, ki sistematično preiskuje prostor stanj (npr. vozlišča v grafu), z namenom najti ciljno stanje. Vozlišča v grafu rekurzivno preiskuje po načelu *„najprej najboljši“* (ang. best first search). Ta metoda na vsakem koraku razvije najbolj obetavno stanje med že razvitimi in še nepregledanimi stanji, ki jih shranjuje v prioritetni vrsti. Trenutno *„najboljše“* stanje algoritem določi s pomočjo hevristične informacije [2,3].

2.1.1. Prostor stanj

Algoritem si dani problem, ki ga rešuje, predstavi s prostorom stanj. Prostor stanj najlažje opišemo z grafom, katerega vozlišča ustrezajo možnim stanjem, medtem ko usmerjene povezave med vozlišči predstavljajo prehode med stanji. Slika 1 prikazuje prostor stanj za problem, ko želimo iz začetnega stanja (točka a) pripotovati do ciljnega stanja (točka *g*). Obisk potnika določenega kraja (na grafu poimenovanega s črko) predstavlja eno stanje.

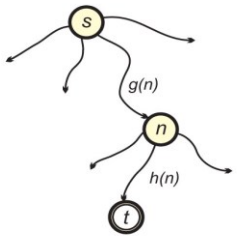


Slika 1: Prostor stanj v obliki usmerjenega grafa (levo) in drevesa možnih poti (desno). Začetno stanje je obarvano z modro, ciljno stanje je obarvano z zeleno.

2.1.2. Hevristična informacija

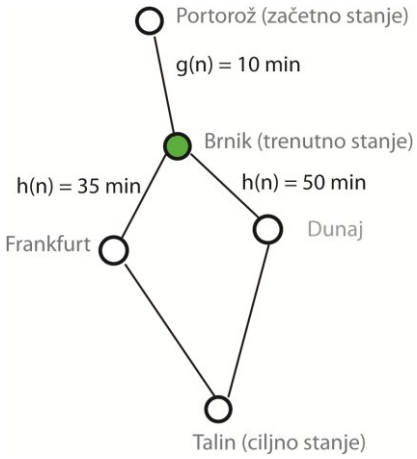
Pri preiskovanju grafa algoritem naleti na več alternativnih možnosti in se o smeri preiskovanja odloča s hevrističnimi ocenami stanj. Hevristična ocena je neka informacija o kvaliteti stanja. Pove nam, kako obetavno je dano stanje pri doseganju cilja.

Algoritem RBFS pri izračunu ocene uporablja cenilko (ang. heuristic estimator) *f(n)*, ki za vsako stanje *n* oceni njegovo obetavnost. Funkcijo *f(n) = g(n) + h(n)* zgradimo kot vsoto dveh členov, kjer *g(n)* predstavlja ceno najboljše poti iz začetnega stanja do stanja *n*, hevristična funkcija *h(n)* pa je ocena najboljše poti od stanja *n* do ciljnega stanja *t*, kot je ponazorjeno na sliki 2.



Slika 2: Hevristična ocena f(n) najcenejše poti od začetnega vozlišča s do ciljnega vozlišča t preko vozlišča n (povzeto po [2]).

Trenutna ocena poti *g(n)* ni nujno najboljša, saj te v procesu iskanja mogoče še nismo odkrili, služi pa nam kot ocena poti od vozlišča *s* do vozlišča *n*. Člen *h(n)* je hevristično ugibanje, saj prostora med vozliščem *n* in ciljnim vozliščem še nismo preiskali [3].
Vzemimo problem izračuna najhitrejše poti letenja iz Portoroža v Talin s pomočjo računalniške simulacije, kot nam ilustrira slika 3.



Slika 3: Prostor stanj za izračun najhitrejše poti letenja letala.

Ko prispemo na brniško letališče, preiskovalni algoritem izračuna točno oceno *g(n)* prepotovane poti od začetnega letališča (Portorož) do trenutnega letališča. Nato lahko nadaljuje pot preko Dunaja ali Frankfurta. Algoritem mora oceniti čas potovanja za oba primera, da se lahko odloči za hitrejšo pot. Ena izmed možnosti je, da algoritem čas letenja med dvema krajema oceni na podlagi zračne razdalje med tema krajema. Na predvidoma krajši poti nas lahko presenetijo

nevihtni oblaki, zaradi varnosti se jim izognemo, ali pa je promet na letališču zelo gost, kar upočasni čas pristajanja in vzletanja. Čas potovanja se tako lahko precej podaljša in zato hevristična ocena *h(n)* ne prikaže realnega stanja. Algoritem pa si z izračunom *h(n)* vseeno pomaga pri odločanju za nadaljevanje poti. Tako bo simulator naprej potoval čez Frankfurt, ker je ta pot ocenjena kot časovno krajša.

2.1.3. Prostorska in časovna zahtevnost RBFS

Za uporabnika sta pomembni tudi prostorska in časovna zahtevnost algoritma. V splošnem je za hevristično preiskovanje večinoma bolj kritična prostorska zahtevnost. Težave s prostorsko zahtevnostjo srečamo pri algoritmu A*, ki ohranja vsa razvita vozlišča v pomnilniku. Za A* velja eksponentna prostorska zahtevnost $O(b^d)$, kjer d označuje dolžino končne rešitve, b pa povprečno število naslednikov razvitih vozlišč. RBFS je nadgradnja algoritma A*, ki optimizira porabo prostora. Tako preidemo iz eksponentnega na linearno (glede na globino iskanja) prostorsko zahtevnost $O(bd)$, saj hranimo samo vozlišča na trenutni iskalni poti in sosednja vozlišča. Posledično se ponovno generirajo določena vozlišča, ki so predhodno že bila generirana, zato govorimo o zmanjšanju prostorske zahtevnosti na račun časovne zahtevnosti. Čas izvajanja je težko oceniti, saj je čas izvedbe odvisen od števila prehodov med potmi pri razvoju vozlišč, kar je odvisno od točnosti hevristične ocene [2, 3].

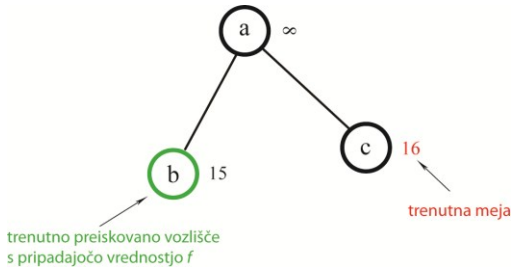
2.2. Implementacija algoritma RBFS

Pri implementaciji algoritma si pomagamo s psevdokodo, prikazano na sliki 4, ki je povzeta po [1].

```
double rbfs(Node searchNode, int bound)
{
   if (searchNode.f > bound)
      return searchNode.f;

   if (goal(searchNode)
      terminate_search(searchNode)

   children = getChildren(searchNode);
   if( children.length == 0)
      return INFINITY;

   for(int i = 0; i < children.length; i++)
  {
      if( searchNode.f < searchNode.fb)
         children[i].fb = max(searchNode.fb, children[i].fb)
      else
         children[i].fb = children[i].f;
   }

   children.sort()//najboljši naj bo vedno na začetku

   while(children[0].fb <= bound)//preverimo najboljšega
   {
      if(children.length == 1)
         fbSibling = INFINITY
      else
         fbSibling = children[1].fb ;
     children[0].fb = rbfs(children[0].fb, min(bound,fbSibling));
     children.sort()//najboljši vedno na začetku
   }
   return  children[0].fb;
}

class Node{

    Object data;
    Node next;

    Node(Object o, Node n){
        data = o;
        next = n;
    }
}
```

Slika 4: Psevdokoda RBFS algoritma.

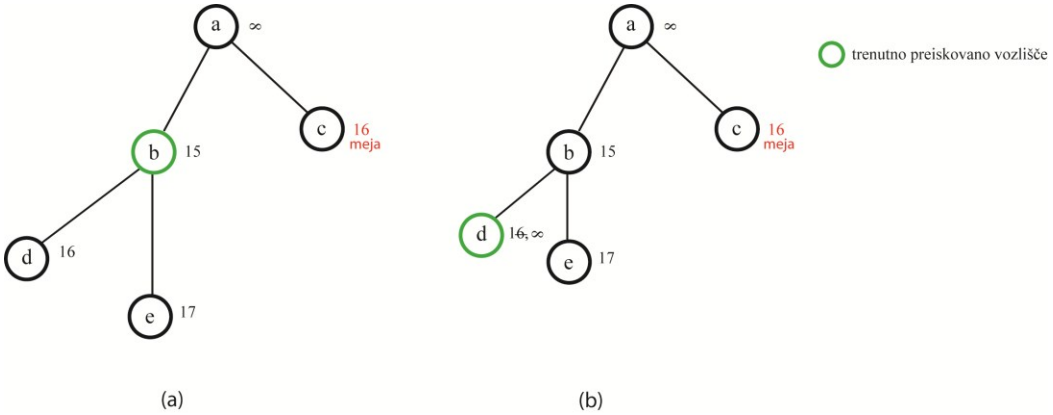
RBFS kličemo rekurzivno z dvema argumentoma: z vozliščem, ki ga hočemo razviti v poddrevo (ang. search node) in z mejo (ang. bound), ki predstavlja zgornjo mejo določeno z vrednostmi *f* sosednjih vozlišč (ang. sibling nodes) [3] vzdolž trenutne iskalne poti, kot to ilustrira slika 5.



Slika 5: Del prostora stanj, ko RBFS sproži rekurzivni klic z dvema argumentoma.

V funkciji najprej preverimo, če je morda vrednost *f* vozlišča večja od meje. V tem primeru se raziskovanje prekine in funkcija vrne vrednost *f* preiskovanega vozlišča. Sledi preverjanje ali je trenutno vozlišče ciljno. Če je tako, se izvajanje algoritma prekine in funkcija vrne končni rezultat.

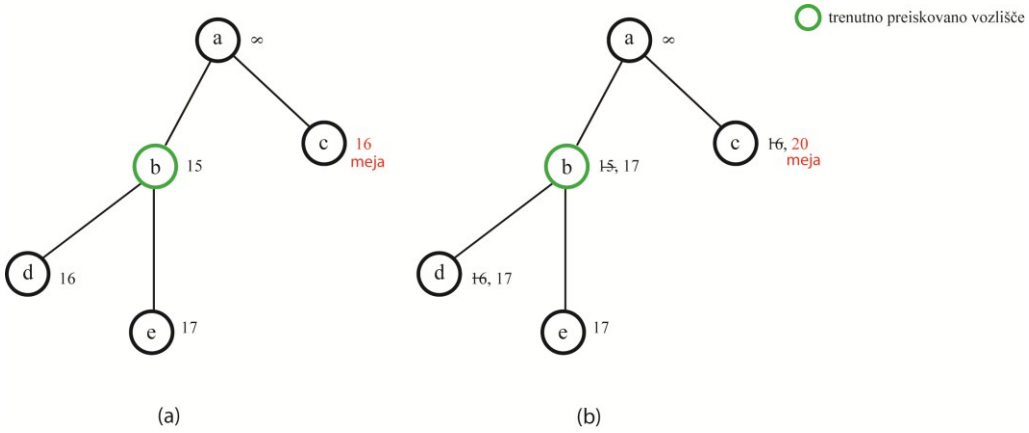
Če omenjena pogoja nista izpolnjena, kličemo funkcijo *getChildren()*, ki vozlišču določi vse naslednike in njihove vrednosti *f*, kot to ilustrira slika 6 na primeru *(a)*. Kadar vozlišče nima naslednikov in ni ciljno vozlišče, dobi vrednost *f* neskončno, kot to prikazuje slika 6 na primeru *(b)*. Z drugimi besedami to pomeni, da po tej veji ni mogoče iskati naprej.



Slika 6: Določanje naslednikov s pripadajočimi vrendostmi f.

Kadar nasledniki obstajajo, vsakemu posebej določimo vrednost *fb*, ki predstavlja posodobljeno vrednost *f* določenega vozlišča. Vrednost *fb* določimo na dva načina (slika 7) :

a) vrednost *fb* naslednika = vrednost *f* naslednika, če vozlišče še ni bilo razvito
b) vrednost *fb* naslednika = *max(*vrednost *fb* korena, vrednost *f* naslednika). To pomeni, da je koren že bil razvit in je bila njegova vrednost *fb* že posodobljena.

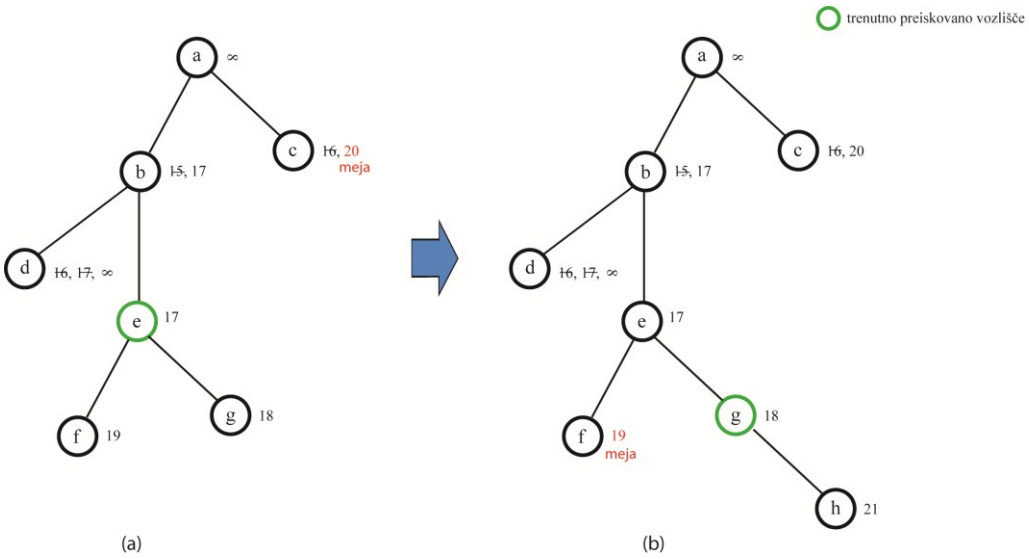


Slika 7: Določitev vrednosti fb naslednikov na dva načina.

RBFS med nasledniki vedno izbere najperspektivnejšega, zato je potrebno tabelo naslednikov urediti v naraščajočem vrstnem redu vrednosti *fb*.

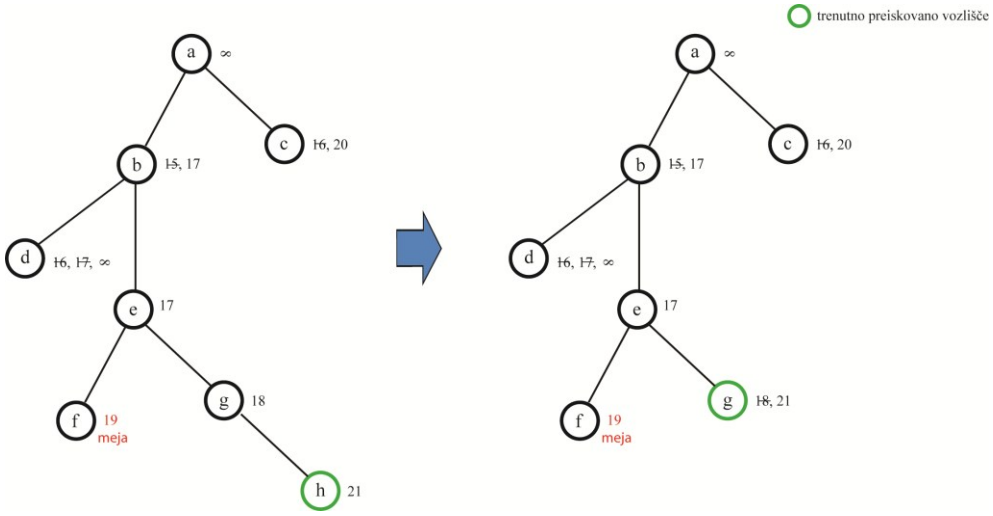
Ko imamo med nasledniki kandidata za preiskovanje (torej je vrednost *fb* kandidata manjša od mejne vrednosti), algoritem vstopi v *while* zanko. V zanki naprej izberemo vozlišče *Sibling*, ki predstavlja najboljšo alternativno pot med nasledniki. Raziskovanje nadaljujemo v podrevesu z rekurzivnim klicem, kjer *searchNode* postane naslednik z najmanjšo vrednostjo *fb* in kjer je meja (drugi parameter rekurzivne funkcije) enaka manjši izmed vrednosti *Sibling* in trenutni meji: *bound = min(bound, fbSibling)* .

Iz enačbe je razvidno, da se meja spremeni, če je vrednost *fb* drugega najperspektivnejšega naslednika (*fbSlibing*) manjša od prvotne mejne vrednosti poddrevesa, kot to ilustrira slika 8.



Slika 8:Zaporedna dela prostora stanj, ko se spremeni vrednost meje.

Pri vračanju rekurzije se vrednost *fb* preiskovanega vozlišča (koren trenutno opuščenega poddrevesa) posodobi z najboljšo *f* vrednostjo naslednikov. Tako lahko RBFS v nadaljevanju odloči, ali je vredno razvijati poddrevo v tem vozlišču. RBFS vozlišča v opuščenem poddrevesu izbriše iz pomnilnika, da prihrani prostor, kot prikazuje slika 9.



Slika 9:Del prostora stanj, ko RBFS izbriše neobetavno poddrevo iz pomnilnika pri vračanju rekurzije.

Tabela naslednikov se ponovno uredi v naraščajočem vrstnem redu vrednosti *fb*. Izvajanje v *while* zanki se nadaljuje, dokler veljata pogoja:

- vrednost *fb* prvega vozlišča iz posodobljene tabele naslednikov je manjša od meje,
- vrednost *fb* prvega vozlišča iz posodobljene tabele naslednikov je končna.

Na koncu algoritem RBFS vrne vrednost *fb* najbolj perspektivnega naslednika raziskovanega vozlišča.

2.3. Prikaz delovanja algoritma RBFS na primeru igre drsečih ploščic

Delovanje algoritma RBFS je prikazano na primeru igre drsečih ploščic. V kvadratu 3x3 imamo osem ploščic označenih s števili od 1 do 8. Ostane nam še eno prosto polje, v katero lahko zdrsne katera koli ploščica, ki meji s praznim poljem. Cilj igre je, da iz katerega koli začetega stanja pridemo do željenega končnega stanja, kot prikazuje slika 10. RBFS poskuša doseči željeno končno stanje z najmanjšim številom premikov ploščic iz začetnega stanja.

(s)

1	3	5
	4	2
6	7	8

(t)

1	2	3
4		5
6	7	8

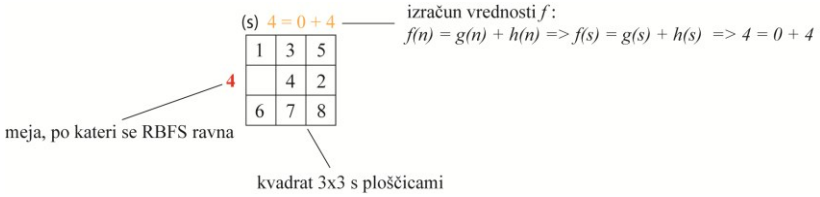
Slika 10: Začetno stanje s in ciljno stanje t za primer igre drsečih ploščic.

V nadaljevanju so s slikami prikazana stanja razvitih vozlišč na trenutni poti, ki so shranjena v pomnilniku med izvajanjem RBFS.

Vsakemu vozlišču določimo vrednost *f*: *f(n) = g(n) + h(n)*, kjer je

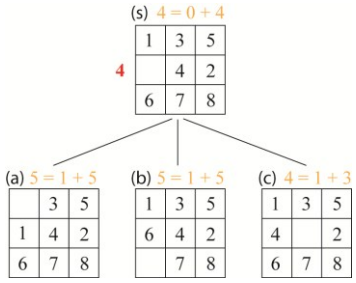
- *g(n)*: število premikov od začetnega stanja do trenutnega stanja,
- *h(n)*: število ploščic, ki niso na ciljni poziciji.

Zaradi preglednosti slik bodo stanja prikazana na način, kot to ilustrira slika 11:



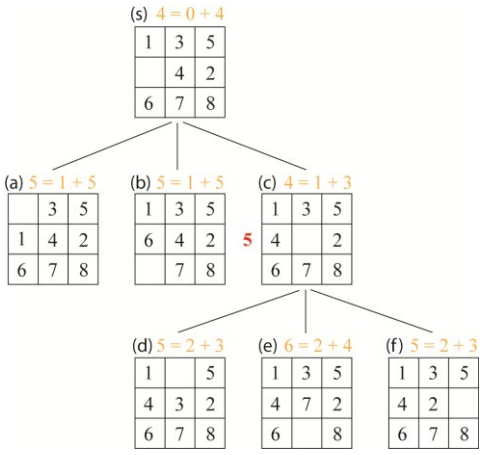
Slika 11:Opis prikazanega stanja.

Algoritem RBFS najprej razvije začetno stanje *s,* prikazano na sliki 12.



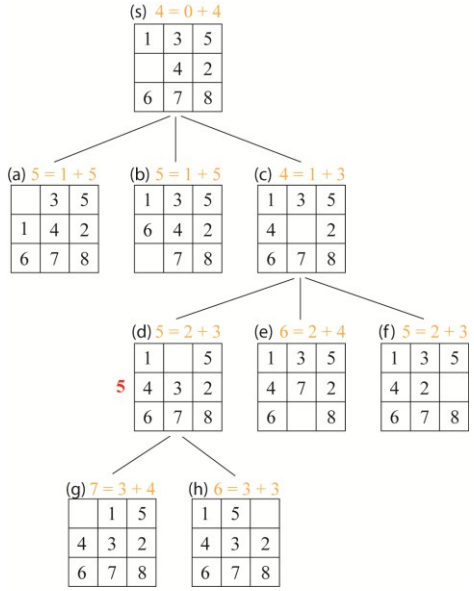
Slika 12: Razvoj prvega stanja (začetnega vozlišča), primer igre drsečih ploščic.

Najperspektivnejši naslednik je vozlišče *c,* zato algoritem naprej razvija poddrevo pod tem vozliščem z mejo 4.



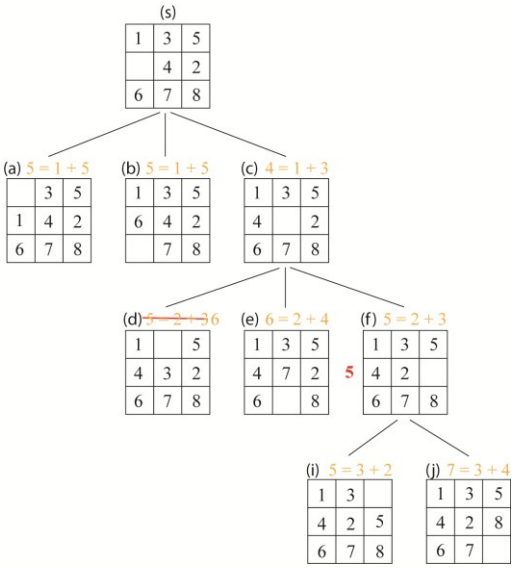
Slika 13: Po razvoju prvega vozlišča RBFS izbere vozlišče c.

Kot prikazano na sliki 14, sledi razvijanje vozlišča *d*, kjer pa vsi nasledniki presežejo mejno vrednost 5. Pot se začasno opusti, vozlišči *g* in *h* se izbrišeta iz pomnilnika. Vrednost *f* vozlišča *d* se posodobi z vrednostjo *f* najbolj pesrpektivnega naslednika, tako ima vozlišče trenutno vrednost *f* enako 6. V tej situaciji je najboljši tekmec vozlišče *f*.

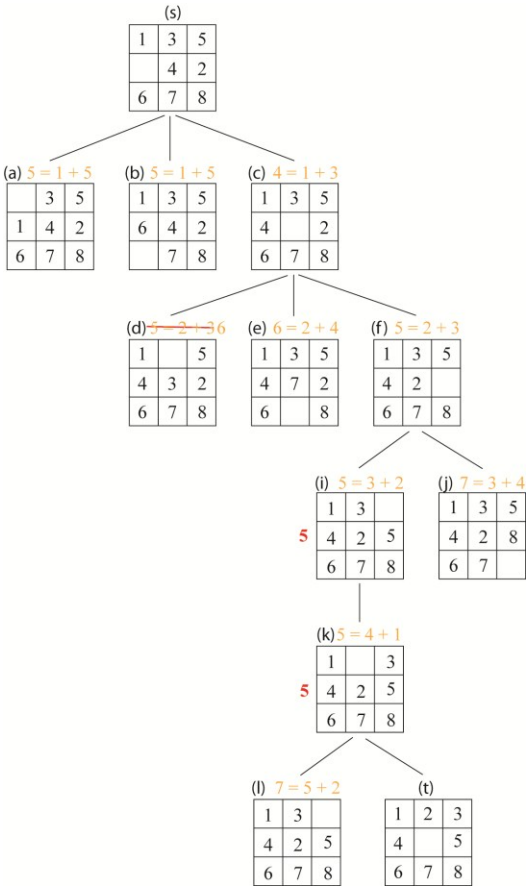


Slika 14: Hevristična ocena naslednikov vozlišča d preseže mejno vrednost, zato algoritem ne nadaljuje po trenutni poti.

Nadaljujemo z razvojem v vozlišču *f*, kot je prikazano na sliki 15. RBFS razvija vozlišča naprej, dokler ne naleti na ciljno vozlišče *t,* prikazano na sliki 16.

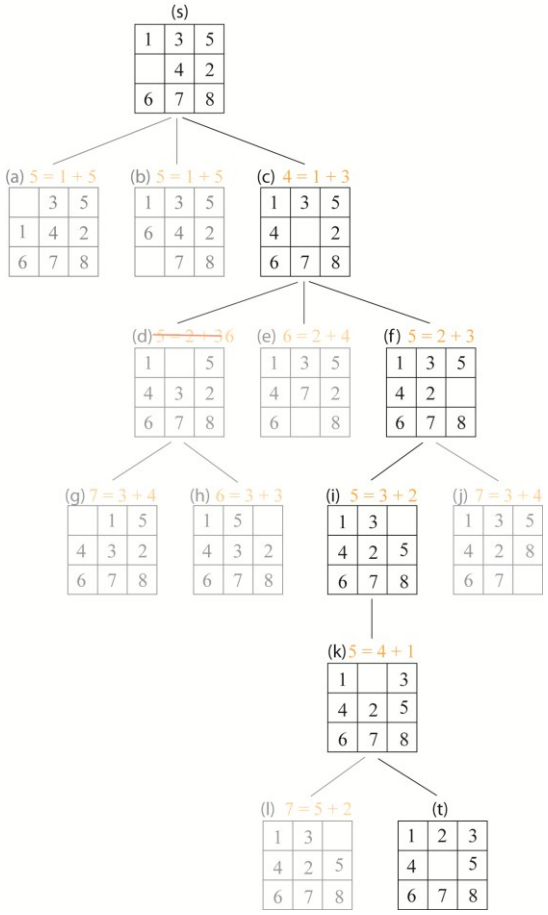


Slika 15: RBFS izbriše naslednike vozlišča d (vrednost f vozlišča d se posodobi z vrednostjo f najperspektivnejšega naslednika) in nadaljuje iskanje pri najboljšem sovozlišču.



Slika 16: RBFS uspešno prispe do ciljnega stanja.

RBFS nam na koncu vrne najkrajšo pot od začetnega stanja *s* do ciljnega stanja *t*, kar je na sliki 17 prikazano s poudarjenimi vozlišči.



Slika 17: Najkrajša pot stanj od začetnega stanja s do ciljnega stanja t.

3. Tehnologija za vizualizacijo in animacijo

3.1. Adobe Flash platforma

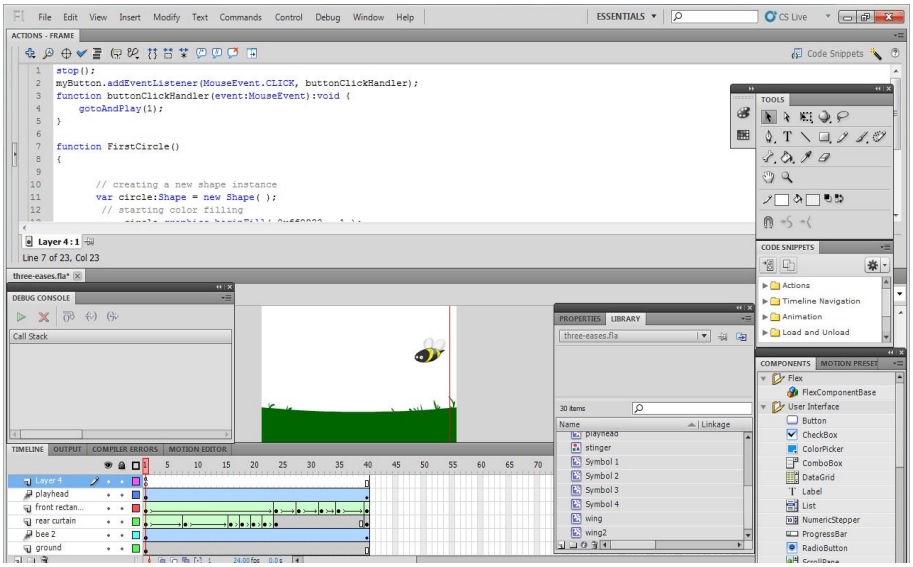
Za razvoj in predstavitev diplomskega dela je uporabljena platforma Adobe Flash (slikovno predstavljena na sliki 18) z razvojnim orodjem Flash Professional in Flash Builder, ter predvajalnikom Adobe Flash Player.

Slika 18: Adobe Flash platforma. Povzeto po [7].

3.1.1. Razvojno orodje Adobe Flash Professional

Adobe Flash Professional je multimedijsko programsko orodje, ki se uporablja za ustvarjanje vsebin za spletne aplikacije, igre, filme, mobilne telefone in druge naprave. Vhodne podatke sprejema preko miške, tipkovnice, kamere ali mikrofona, prav tako pa podpira dvosmerno pretakanje video in avdio podatkov (ang. audio and video streaming).
Flash razvojno okolje ponuja [11]:

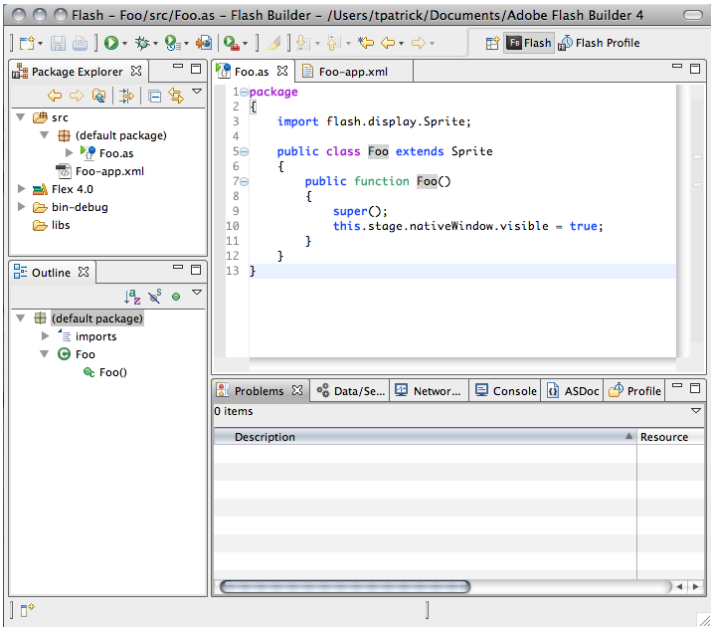
- orodja za razvoj in oblikovanje internetnih aplikacij,
- napredna orodja za interaktivne in video projekte,
- bogate risane in animirane produkte; vektorska grafika omogoča, da se lahko velikost uporabljenih grafičnih elementov poljubno spreminja, brez izgube kvalitete in ločljivosti,
- razvoj aplikacij v jeziku ActionScript v urejevalniku kode s solidnim razhroščevalnikom,
- hitro vključitev programskega jezika ActionScript za različne namene, npr. navigacija časovne vrstice, manipulacija akcij in animacij, proženje efektov, predvajanje zvoka in videa, obdelava dogodkov (ang. *Event Handler*) itd.
- integracijo komponent grafičnih orodij kot so Photoshop, Gimp, Illustrator, CorelDraw, InDesign in podobno,
- objavo programov na napravah, ki uporabljajo Adobe AIR za namizne aplikacije in razširjen Adobe Flash Player integriran v internetne brskalnike.



Slika 19: Vpogled v razvojno orodje Flash Professional.

3.1.2. Razvojno orodje Adobe Flash Builder

Adobe Flash Builder (prej poznan kot Adobe Flex Builder) je razvojno okolje (ang. integrated development environment IDE), razvito na Eclipse platformi za razvoj bogatih internetnih (poganjamo s Flash Player-jem) in namiznih (poganjamo z AIR programsko opremo) aplikacij. Vključuje podporo inteligentnemu kodiranju, razhroščevanju, vizualnemu oblikovanju in testiranju naših projektov. Orodje vsebuje urejevalnike predvsem za MXML in ActionScript programska jezika [23, 24].

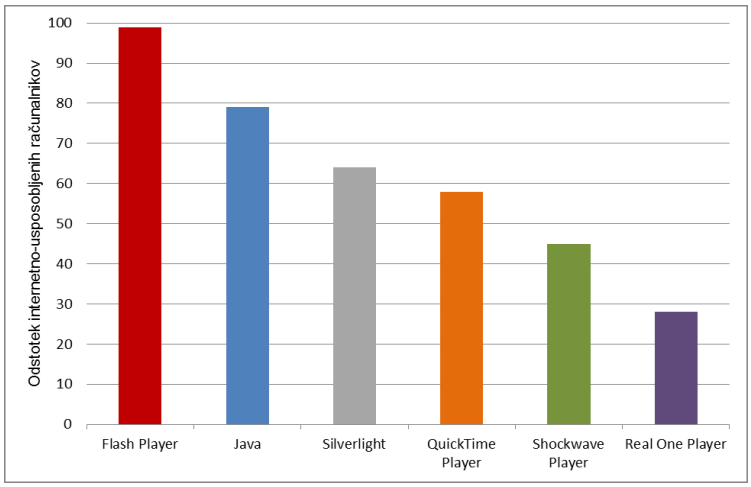


Slika 20: Vpogled v razvojno orodje Flash Builder.

Za marsikoga (predvsem za začetne ali občasne ActionScript razvijalce) orodje Flash predstavlja prevelik finančni zalogaj, zato omenimo še alternativna odprto kodna orodja; npr. FlashDevelop, ki je brezplačen in prav tako na visoki ravni.

3.1.3. Adobe Flash Player - programska platforma odjemalcev

Končni prodkut je dateteka tipa SWF, ki jo predvajamo na odjemalcu Adobe Flash Player. Flash Player je na voljo kot priključek (ang. plugin) za različne verzije internetnih iskalnikov (na primer Mozilla Firefox, SeaMonkey, Opera, Safari, Internet Explorer, Chrome (integriran)) na različnih platformah (Windows, Linux, Solaris, Mac OS X, Android). Ta predvajalnik smo izbrali, ker ga po raziskavi Millward Brown in Rich Internet Application Statistics (narejeni na zrelih trgih, ki zajemajo: ZDA, Kanado, Veliko Britanijo, Francijo, Nemčijo, Japonsko, Avstralijo) uporablja preko 3 milijone razvijalcev. Teče na 99% računalnikov z dostopom do interneta (glej graf 1) ter na široki paleti ostalih mobilnih naprav (več o raziskavi in statistiki v [7,21,22]). Tako večina odjemalcev nima potrebe po dodatnih namestitvah programske opreme.



Graf 1: Delež multimedijskih odjemalcev na računalnikih z dostopom do interneta [6,22].

3.1.4. Uporaba rešitev Adobe Flash platforme v praksi

Za lažjo predstavo je navedenih nekaj praktičnih primerov uporabe Adobe Flash platforme:

- The New York Times – digitalno založništvo.
  Produkt Times Reader omogoča fleksibilno objavljanje člankov in drugih novinarskih vsebin v digitalni obliki. Prav tako končnim uporabnikom ponuja zanimivo interakivno programsko okolje za prebiranje vsebin na različnih operacijskih sistemih.
- Playfish - razvijanje iger z več igralci za socialna omrežja.
  Podjetje Playfish razvija družabne igre, med katerimi je tudi pet od desetih najbolj priljubljenih iger na Facebooku. Igre vključuje 3D grafiko, prilagoditev oseb in igre,

namenjene več igralcem. Flash platforma omogoča hiter razvoj in podpira integracijo s Facebookom, MySpaceom in drugimi družabnimi omrežji [18].

- NASDAQ - obširne analize podatkov z bogato in dinamično predstavitvijo,
- NATO - podporni sistem za misije,
- Sony Ericsson – aplikacije za mobilne telefone,
- DirecTV - dostop do živih videoposnetkov in prenos podatkov do različnih plaform,
- SAP - enostavna in bogata vizualizacija podatkov.

3.1.5. Slabosti Adobe Flash platforme

Podobno kot ostale tehnologije ima Flash platforma poleg prednosti tudi nekatere slabosti. Omejimo se na tiste, ki so pomembne pri razvoju in uporabi spletne aplikacije.
Vsebine, predvajane predvsem s starejšimi verzijami Flash predvajalnikov, zavzamejo velik del delovnega procesorja, še posebno takrat, ko animacije vsebujejo veliko grafičnih objektov in efektov. Glavni procesor prevzame večji del nalog za izris grafike, ker Flash predvajalnik vstavljen v internetni brskalnik nima direktnega dostopa do procesorja grafične kartice. Občasno lahko pride tudi do nepričakovanih napak v izvajanju aplikacij, kar pripelje celo do sesutja Flash predvajalnika ali celo brskalnika. Pojavljajo se tudi varnostne luknje, ki ogrožajo varnost uporabnika spletnih aplikacij. Večje težave imajo tudi spletni iskalniki, ki največkrat niso sposobni razbrati vsebine Flash datotek. Tako težje najdemo pot do želene Flash vsebine na internetu. Za razvijalce je največja težava Flash platforme v tem, da gre za licenčno tehnologijo, kjer so razvojna orodja dostopna le za plačilo [31,32].

3.2. Druge alternativne tehnologije primerne za vizualizacijo in animacijo

Navedimo nekaj alternativnih tehnologij, ki so prav tako primerne za vizualizacijo in animacijo algoritmov ter drugih multimedijskih vsebin.

3.2.1. Microsoft Silverlight

Silverlight je multimedijska platforma, zelo podobna Flash platformi. Prikazuje lahko video, audio in interaktivne animacije, ki so vgrajene v internetne strani. Platformo razvija Microsoft in je kompatibilna z Windows, Mac OS X in Linuxom (kompatibilnost omogoča program Moonlight) [21].

3.2.2. Java

Java apleti se uporabljajo za interaktivne predstavitve, video, in 3D predstavitve. Uporabljajo je za bolj zahtevne vizualizacije, podprte z visokonivojskim porgramiranjem in komunikacijo med odjemalcem in strežnikom. Sun je razvil produkt JavaFX, ki je postal močna alternativa ostalim bogatim internetnim aplikacijam RIA (ang. Rich Internet Application) [25, 27].

3.2.3. DHTML

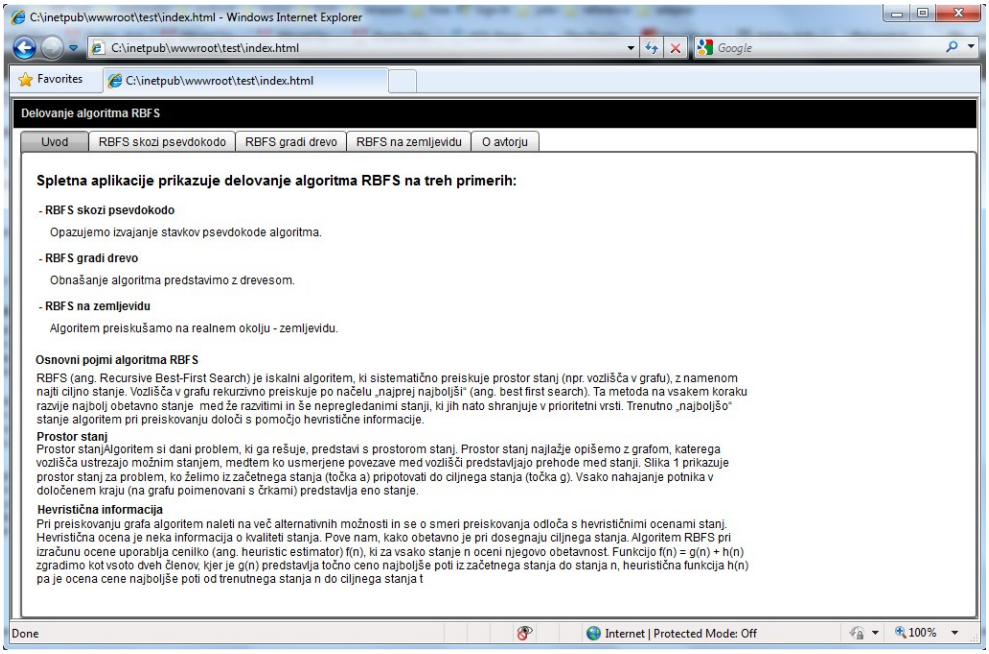
Pri dinamični spletni strani (DHTML) lahko uporabimo več različnih, med seboj povezanih tehnologij, npr. statični označevalni jezik, kot je HTML, scripni jezik JavaScript, vizualno opredelitveni jezik CSS in podobno.

3.2.4. HTML 5

HTML5 je naslednja velika prenova HTML standarda. Dodanih je veliko novih funkcij, ki vsebujejo video in audio. Integriran je tudi standard vektorske grafike SVG (ang. Scalable Vector Graphics). Vse to omogoča vključevanje in manipuliranje z multimedijskimi vsebinami na internetnih straneh brez vtičnikov in ostalih aplikacij.

4. Predstavitev spletne aplikacije »*Delovanje algoritma RBFS*«

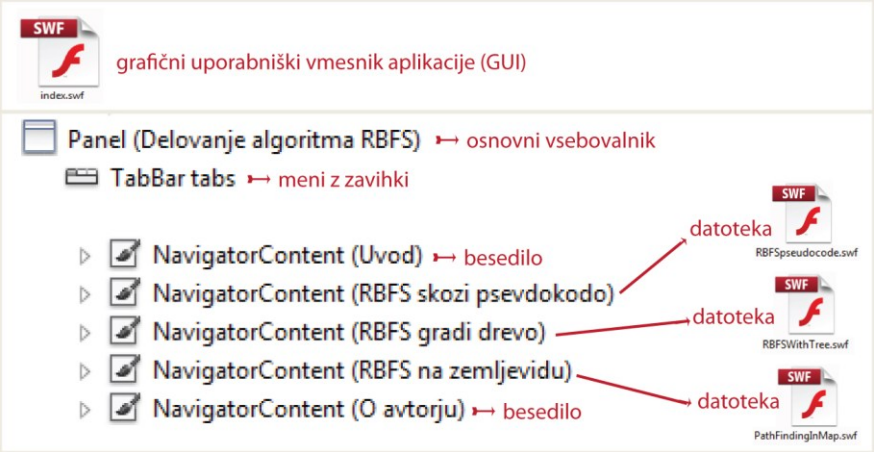
V internetnem brskalniku z nameščenim vtičnikom Flash Player odpremo aplikacijo z datoteko *index.html*, kjer predvajalnik odpre vgnezdeno datoteko *index.swf*, kot je prikazano na sliki 21.



Slika 21: Aplikacijo zaženemo v internetnem brskalniku.

Prikaže se osnovni grafični uporabniški vmesnik (GUI) aplikacije, Z vsebino prikazano na sliki 22. Na osnovni vsebovalnik dodamo meni z zavihki, da lahko preklapljamo med vsebinami:

- *Uvod*: kratek tekstovni opis vsebine aplikacije in opis algoritma RBFS.
- *RBFS skozi psevdokodo*: z datoteko *RBFSpseudocode.swf* predstavimo delovanje algoritma skozi psevdokodo.
- *RBFS gradi drevo*: z datoteko *RBFSWithTree.swf* delovanje RBFS opazujemo na zgrajenem drevesu.
- *RBFS na zemljevidu*: datoteka *PathFindingInMap.swf* prikazuje kako RBFS išče najkrajšo pot na zemljevidu.
- *O avtorju*: kratek tekstovni opis avtorja aplikacije.



Slika 22: Grafični uporabniški vmesnik aplikacije.

4.1. Predstavitev primera »RBFS skozi psevdokodo«

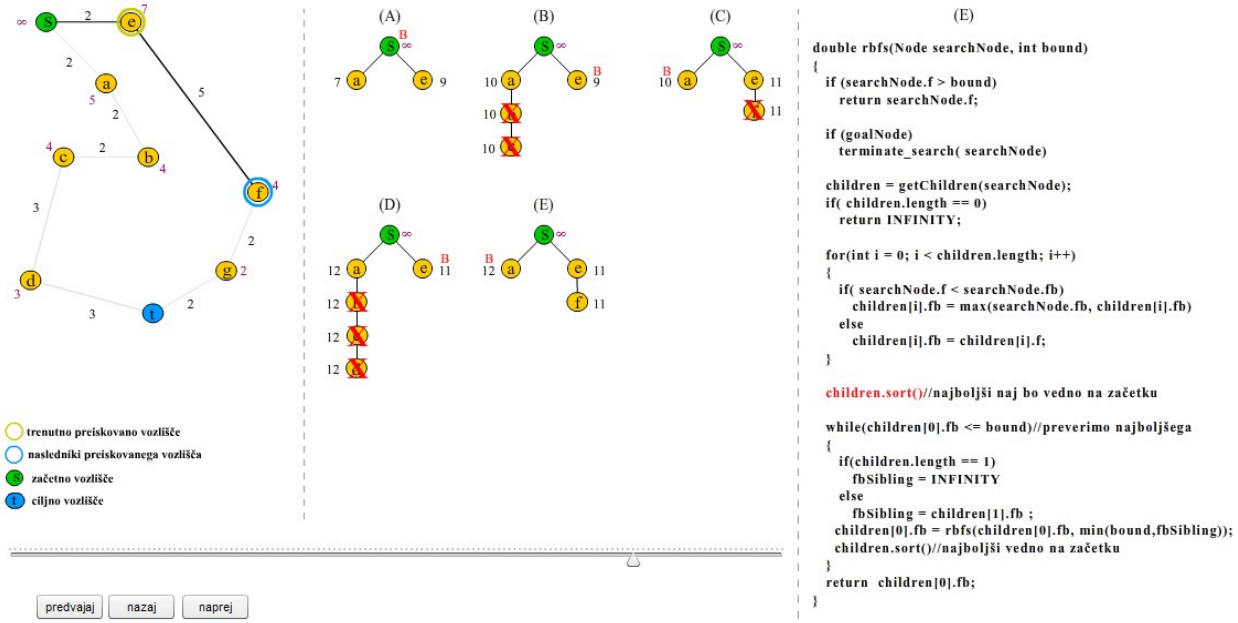
Z animacijo uporabnik spremlja obnašanje algoritma RBFS preko izvajanja ukazov v psevdokodi, vzporedno pa opazuje dogajanje na preprostem grafu in stanje v pomnilniku. Animacija omogoča koračno sledenje od enega do drugega stavka v psevdokodi, podobno kot to omogoča razhroščevalnik. Tako je omogočen podroben vpogled v delovanje algoritma.

4.1.1. Struktura primera »RBFS skozi psevdokodo«

Animacija reševanja problema iskanja najkrajše poti od točke s do točke t je sestavljena iz treh delov, kot je prikazano na sliki 23 (primer povzet po [3]) . Skrajno levo prikazujemo delovanje RBFS na preprostem grafu. Vozlišče predstavlja kraj. Vijolične številke poleg vozlišč predstavljajo vrednosti funkcije h(n) za vozlišče, črne številke na povezavah predstavljajo vrednosti g(n). Zaradi lažjega sledenja se med animacijo trenutno preiskovano vozlišče obarva z zelenim obročem, nasledniki pa so obarvani z modro.

Na sredini prikazujemo razvita vozlišča na trenutni poti, ki so hranjena v pomnilniku. Ko se zgodi večja sprememba, posnetka ne izbrišemo, kot to dejansko naredi RBFS, ko neperspektivno poddrevo izbriše iz pomnilnika. V animaciji taka podrevesa prekrižamo z rdečim znakom X. Tako so vidna vsa dosedanja stanja v pomnilniku (vidimo sled algoritma) in uporabnik ima širši pogled nad dogajanjem. Vrednost *f* pod rdečim *B* predstavlja mejo določenega stanja.

Skrajno desno spremljamo delovanje algoritma skozi psevdokodo, kot je predstavljeno v poglavju 2.2, ko se vrstica obarva rdeče. To predstavlja izvedbo ukaza v algoritmu, posledice ukazov pa vidimo v vizualizaciji grafa.

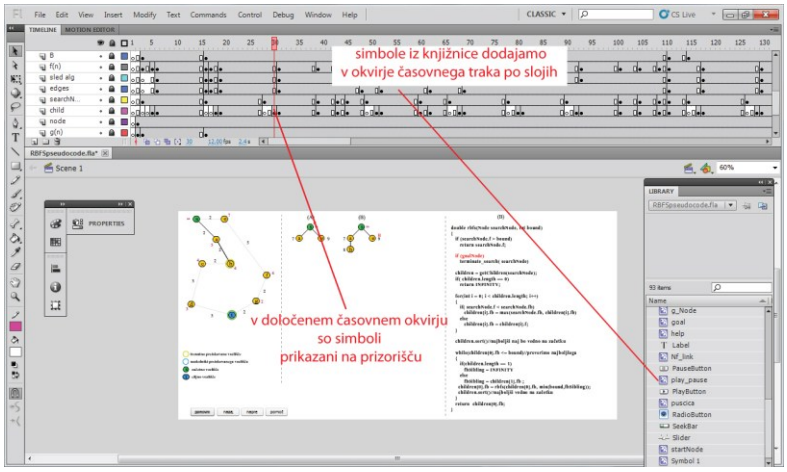


Slika 23: Izgled primera »RBFS skozi psevdokodo«.

Uporabnik požene animacijo z gumbom »predvajaj«. S pritiskom na gumb »premor« ustavimo animacijo. Gumba »naprej« in »nazaj« omogočata spremljanje animacije korak za korakom. Dodatne informacije se prikažejo s pritiskom na »pomoč«. S pomočjo drsnika se časovno orientiramo in premaknemo na željeno stanje algoritma.

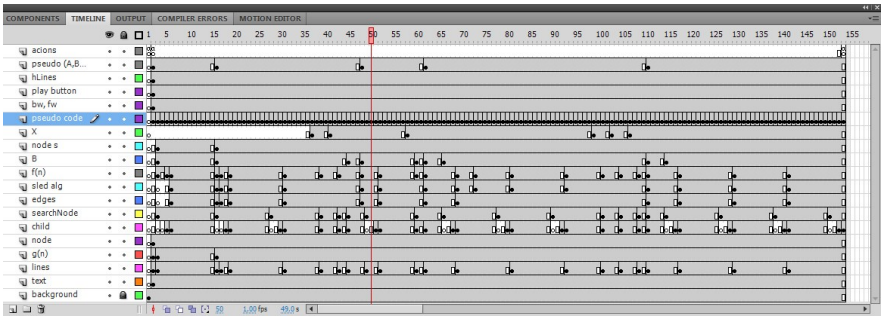
4.1.2. Implementacija primera »RBFS skozi psevdokodo«

Primer »*RBFS skozi psevdokodo«* smo razvili kot animiran film v Flash okolju in ga izvozili v datoteko *RBFSpseudocode.swf*. Ko odpremo nov dokument v Flash okolju, se ustvari kvadratno prozorišče s časovnim trakom. Najprej ustvarimo ali uvozimo grafične elemente, ki jih shranimo kot simbole v knjižnico. Simboli so objekti tipa *MovieClip*, *Button* ali *BitMap.* Te simbole in ostale privzete grafične komponente orodja Flash postavljamo na prizorišče, kot prikazuje slika 24. Vsak simbol postavimo v nov sloj zaradi morebitnega prekrivanja med animacijo.



Slika 24: Dodajanje simbolov na prizorišče Flash orodja.

Psevdokodo dodamo kot tekstovni objekt. Vsak programski stavek v psevdokodi pripada enemu okvirju časovnega traka, kot je prikazano na sliki 25. Tako lahko po korakih sledimo izvajanju programskih stavkov. Spremembe na grafu in posnetkih stanj v pomnilniku vzporedno sledijo ukazom iz psevdokode. Sprememba, ki je posledica izvedbe programskega stavka, je umeščena v isti časovni okvir.



Slika 25:Ustvarjanje časovnih okvirjev v časovnem traku.

Poleg psevdokode na osnovni gradnik dodamo še graf krajev in drevesa stanj, ki so sestavljeni iz objektov *vozlišče* in *povezava* tipa *MovieClip*.

Objekt *vozlišče* je sestavljen iz treh slojev, kot je prikazano na sliki 26. Sloj *grafika* predstavlja obarvan krog, sloj *ime* poimenuje vozlišče, sloj *h(n)* prikaže vrednost *h(n)* vozlišča.

Objekt *povezava* je sestavljen iz dveh slojev, kot je prikazano na sliki 26. Sloj *črta* grafično prikaže povezavo med dvema vozliščema, sloj *g(n)* pa prikaže številčno vrednost prehoda iz enega vozlišča v drugo.

Vsak sloj vsebuje grafični ali tekstovni element, ki mu lahko dinamično spreminjamo barvo, ime in vrednost *h(n)* ali *g(n)*.

Slika 26: Struktura gradnika vozlišče (levo) in povezava (desno).

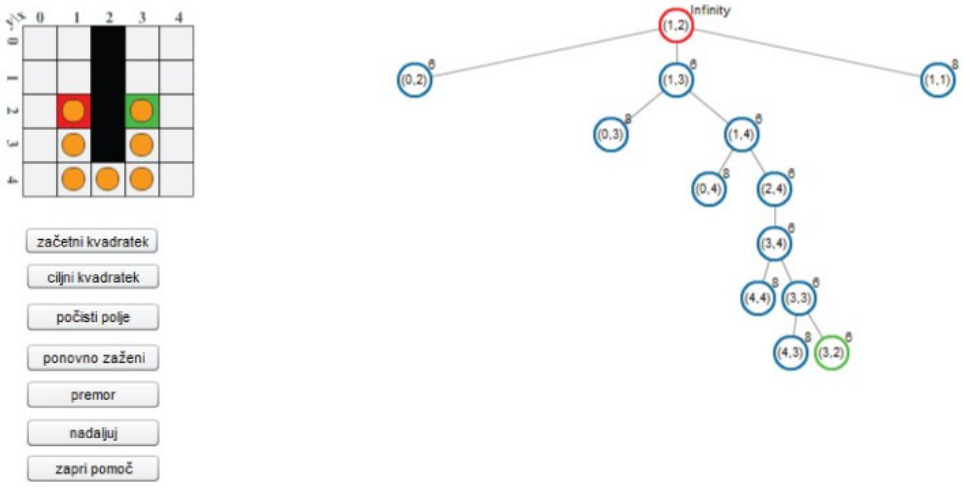
4.2. Predstavitev primera »RBFS gradi drevo«

V prejšnjem poglavju smo se na enostavnem primeru podrobneje seznanili, kako se izvaja RBFS. V primeru *»RBFS generira drevo«* pa opazujemo obnašanje algoritma RBFS v različnih situacijah, na katere lahko sami vplivamo. RBFS v tem primeru išče najkrajšo pot med začetnim kvadratkom in končnim kvadratkom v dvodimenzionalnem polju velikosti 5x5. Prehodi so možni v vse smeri razen diagonalno.

4.2.1. Struktura primera »RBFS gradi drevo«

Vidno polje animacije je razdeljeno na dva dela, kot to prikazuje slika 27:

- levo vidimo prikaz polja in gumbov za proženje akcij,
- desno vidimo prikaz drevesa, ki ga gradi RBFS.



Slika 27: Vizualni izgled primera »RBFS gradi drevo«.

Na grafičnem vmesniku se nahajajo gumbi:

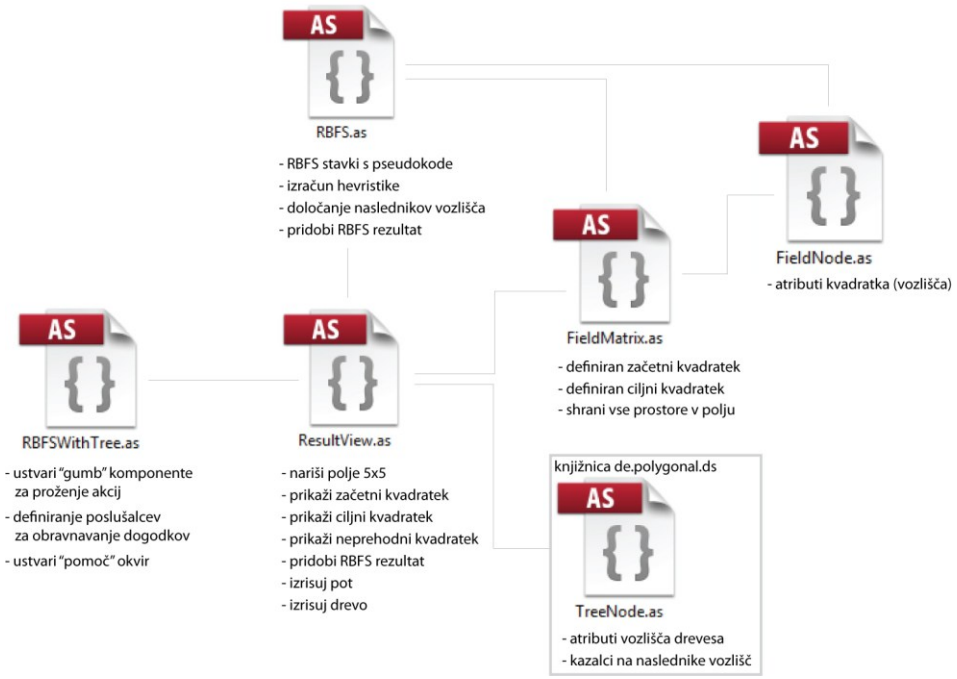
- »začetni kvadratek« - po aktivaciji bomo s prvim klikom na polje določili začetni kvadratek.
- »ciljni kvadratek« - po aktivaciji bomo s prvim klikom na polje določili ciljni kvadratek.

- »počisti polje« - polje se vzpostavi v začetno stanje.
- »ponovno zaženi« - ponovno zaženemo animacijo algoritma na trenutnem polju.
- »premor« - začasno ustavi animacijo.
- »nadaljuj« - nadaljujemo z animacijo.
- »pomoč« - prikličemo okno z dodatnimi informacijami.

Na polju določimo začetni kvadratek (obarvan rdeče) in ciljni kvadratek (obarvan zeleno). Ko sta omenjena kvadratka definirana, se animacija delovanja RBFS samodejno sproži. Uporabnik lahko po korakih spremlja, kako RBFS preiskuje polje in išče najkrajšo pot (rumeni krogi v polju) do cilja. S klikom na polje lahko določamo tudi neprehodne kvadratke (obarvani s črno). Omogočeno je sprotno spremljanje generiranja in preiskovanja vozlišč, prikazanih v obliki drevesa, ki ga gradi RBFS. Vozlišča drevesa predstavljajo kvadratke v našem polju. Pozicijo kvadratka lahko razberemo z vrednosti *x* in *y* znotraj vozlišča, ki so ekvivaletne vrednostim *x* in *y* na polju. Za primer vzemimo začetni kvadratek, ki se v polju nahaja na poziciji *(1,2)*. To vrednost razberemo iz začetnega vozlišča drevesa. Poleg vozlišča se nahaja vrednost *f* vozlišča, ki se pri vračanju rekurzije posodobi.

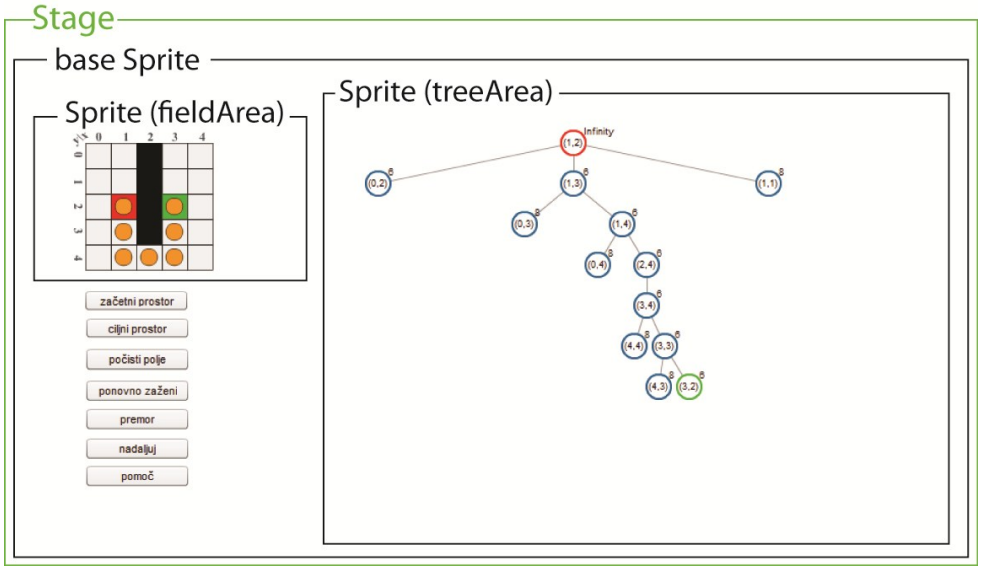
4.2.2. Implementacija primera »RBFS gradi drevo«

»*RBFS gradi drevo*« izvozimo v datoteko *RBFSWithTree.swf*. Programsko arhitekturo in implementacijo razredov razložimo s pomočjo diagrama razredov na sliki 28.



Slika 28: Arhitektura razredov za primer »RBFS gradi drevo«.

Ko se datoteka *RBFSWithTree.swf* odpre v predvajalniku, se pokliče kostruktor glavnega razreda *RBFSWithTree.* Ustvari se osnovna instanca vsebovalnika tipa *Sprite*, ki je dodana na prizorišče tipa *Stage,* kot je ponazorjeno na sliki 29. Prizorišče si lahko predstavljamo kot celotno vidno polje animacije.

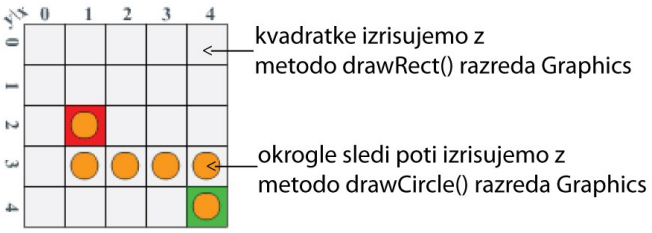


Slika 29: Sestava grafičnega uporabniškega vmesnika za primer »RBFS gradi drevo«.

Na osnovni vsebovalnik dodamo grafične objekte tipa *SimpleButton*, jim določimo lastnosti, registriramo poslušalce dogodkov ter implementiramo metode za odzivanje. V glavnem razredu kličemo konstruktor razreda *ResultView*, v katerem ustvarimo še dva vsebovalnika *fieldArea* in *treeArea*, ki sta tipa *Sprite*.

Izris polja kvadratkov

Na vsebovalniku *fieldArea* prikazujemo animacijo iskanja najkrajše poti od začetnega kvadratka do ciljnega kvadratka na polju. Polje (množica kvadratkov) je objekt tipa *FieldMatrix*, kjer definiramo atribute polja (višina, širina) in ustvarimo dvodimenzionalno tabelo kvadratkov, ki so tipa *FieldNode.* V razredu *FieldNode* so definirani atributi (prehodnost, začetni kvadratek, ciljni kvadratek, pozicija, vrednosti hevristke, število naslednikov) posameznega kvadratka v polju. Kvadratke izrisujemo z grafičnimi objekti tipa *Graphics,* kjer metoda *drawRect()* izriše kvadrate in jih ustrezno obarva glede na značilnosti (prehoden, neprehoden, začeten, ciljni), ki jih določi uporabnik.



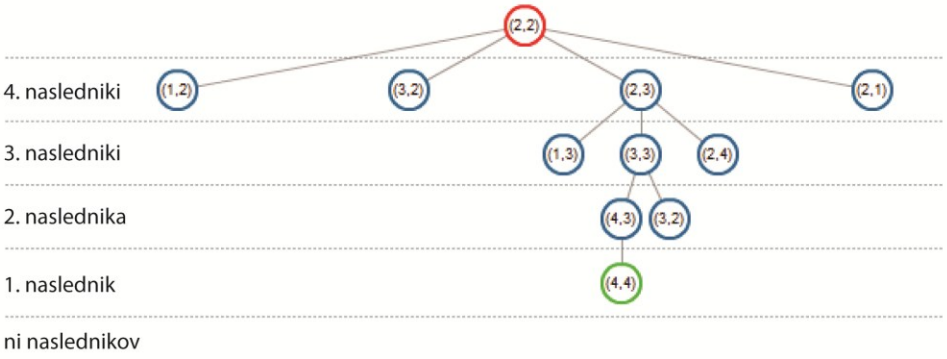
Slika 30: Izrisovanje polja kvadratkov in poti preiskovanja.

Izris poti preiskovanja

Rezultat algoritma RBFS prikažemo z rumenimi krogi, ki jih izriše metoda *drawCircle()*, kot nam prikazuje slika 30. Podatke za izris poti na polju nam posreduje razred *RBFS*, kjer so implementirani programski stavki algoritma RBFS povzeti po psevdokodi, ki smo jo predstavili v poglavju 2.2. Vse preiskovane kvadratke (vozlišča) shranimo v vrsto *_pathNodes*.

Določanje naslednikov

Vsakemu kvadratku potencialno pripadajo nasledniki, zato jih je potrebno določiti z metodo *getChilds() v* razredu *RBFS*. Če preiskovani kvadratek na vseh straneh meji na prehodne kvadratke, mu dodelimo največ štiri naslednike (levo, desno, gor, dol), v drugih primerih pa ustrezno manj. Primeri določanja naslednikov so prikazani na sliki 31.



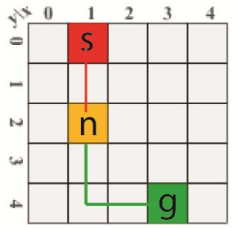
Slika 31: Določanje naslednikov preiskovanih kvadratkov.

Izračun hevristike

Da RBFS izbere najperspektivnejšega naslednika, je potrebno izračunati funkcijo *f(n) = g(n) + h(n.)* Funkcijo *g(n)* določimo s številom korakov (na že preiskovani poti) od začetnega kvadratka do trenutnega kvadratka *n*. Hevristično oceno kvadratka *h(n)* določi metoda *heuristic()* v razredu *RBFS*.

```
//manhattan heuristic
private function heuristic(node:FieldNode):Number
{
        return Math.abs(node.x - goalNode.x) + Math.abs(node.y - goalNode.y);
}
```

Funkcijo *h(n)* določimo z manhattansko razdaljo, ki je definirana kot vsota razdalj med dvema točkama na pravokotnih oseh *x* in *y*, kot na primeru prikazuje slika 32. Enačba za razdaljo med točkama $N = \{x_1, y_1\}$ in $G = \{x_2, y_2\}$ se glasi: $D = |x_1 - x_2| + |y_1 - y_2|$.



$f(n) = g(n) + h(n) = 2 + 4 = 6$

$h(n)_{manhattan} = |x_n - x_g| + |y_n - y_g| = 2 + 2 = 4$

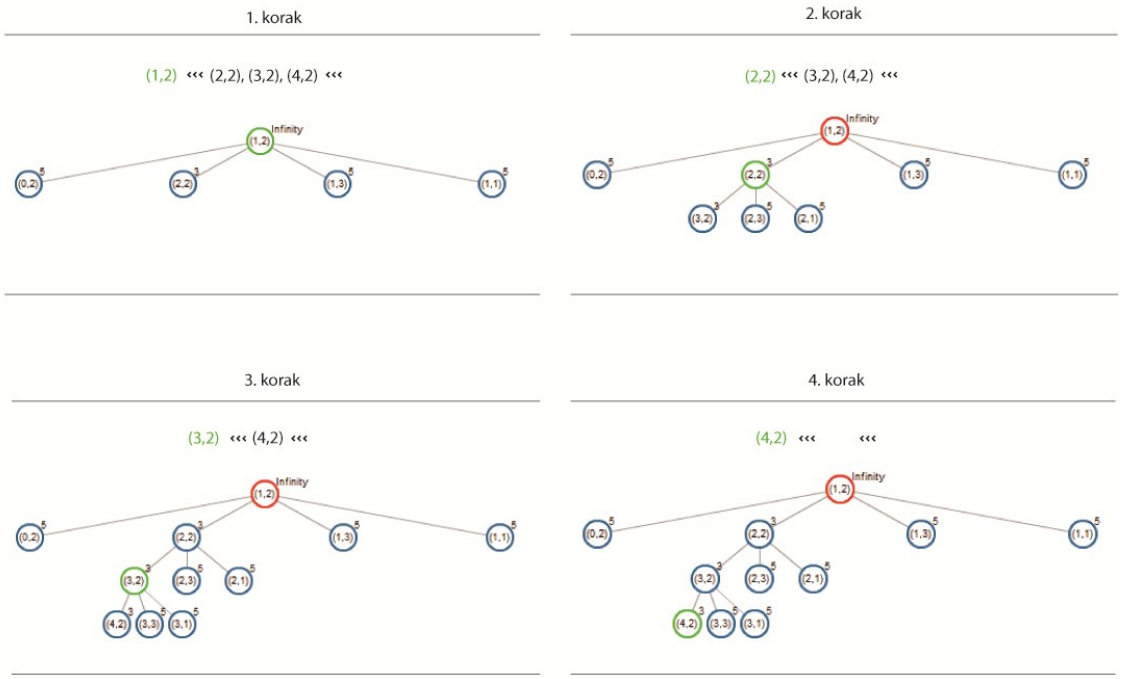
Slika 32: Izračun cenilke f na za kvadrratek n v polju.

Izris drevesa

Na vsebovalniku *treeArea* prikazujemo izgradnjo drevesa. RBFS algoritem vsak preiskan prostor shrani v podatkovno strukturo *vrsta*, iz katere v metodi *createTree()* razreda *ResultView* črpamo podatke in gradimo drevo. Vozlišče drevesa je objekt tipa *TreeNode* (knjižnica ActionScript razredov podatkovnih struktur je povzeta po [30]). Konstruktor razreda TreeNode generira nov kazalec, ki kaže na predhodnika. Na ta način se lahko vračamo po drevesu navzgor in porežemo poddrevo naslednikov, podobno kot algoritem RBFS iz spomina izbriše neperspektivna poddrevesa. Naslednike vozlišča shranimo v podatkovno strukturo *dvojno povezani seznam*, zato se s pomočjo iteratorja lahko sprehajamo po seznamu naslednikov naprej in nazaj.

Ko v podatkovno strukturo *drevo* dodamo vozlišča ali odstranimo poddrevo, se sprememba takoj izriše na vsebovalnik grafičnih elementov, kot prikazuje slika 33. Tako uporabnik po korakih spremlja delovanje algoritma. Vozlišče drevesa v obliki kroga izrišemo z metodo *drawCircle()* razreda *Graphics*. Atribute vozlišča (pozicija na polju, vrednost *f*) predstavimo s tekstovnimi

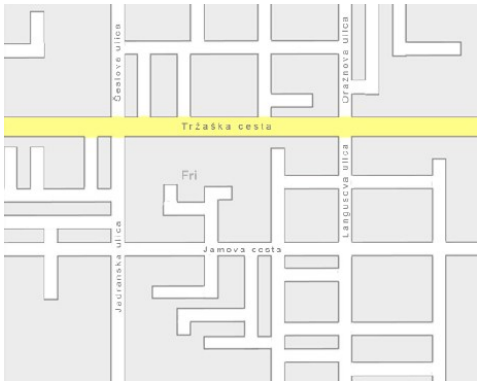
objekti tipa *TextField*. Povezave med vozlišči izrišemo z metodo *lineTo()* razreda *Graphics*. Z rekurzivnim klicem metode *drawTree()* razreda *ResultView* najprej rezerviramo pozicijo posameznega vozlišča (glede na globino vozlišča in število sorodnikov), ob vračanju rekurzije pa izrisujemo grafične elemente.



Slika 33: Pridobivanje podatkov iz podatkovne strukture vrsta za izgradnjo drevesa.

4.3. Predstavitev primera »RBFS na zemljevidu«

Primer *»RBFS na zemljevidu«* predstavlja, kako RBFS reši problem iz realnega okolja. Poskušamo najti najkrajšo pot med začetno točko in ciljno točko na delu zemljevida mesta Ljubljana, ki ga ilustrira slika 34.



Slika 34: Zemljevid okolice FRI v Ljubljani.

4.3.1. Struktura primera »RBFS na zemljevidu«

Na vidnem polju animacije se nahaja zemljevid (slika 35). Določimo začetno točko, kjer se potovanje prične (rdeč kvadrat), in ciljno točko, kamor smo namenjeni (zelen kvadrat). Na zemljevidu lahko označujemo ovire na cesti (črn kvadrat). Najkrajšo pot med točkama izrišemo z oranžnimi krogi.

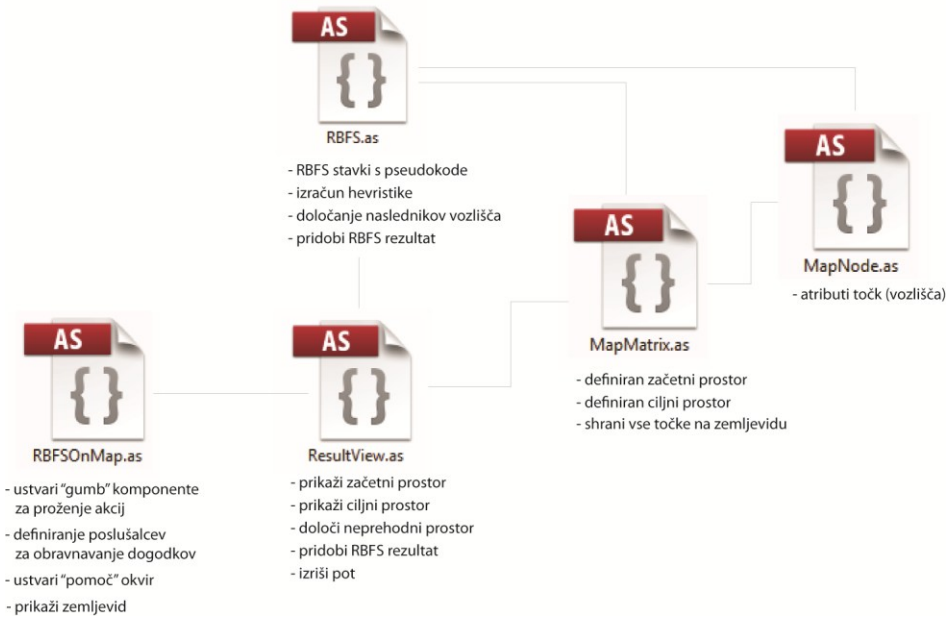
Slika 35: Vizualni izgled primera »RBFS na zemljevidu«.

Poleg zemljevida se nahajajo naslednji gumbi:

- »začetna točka« - po aktivaciji gumba bomo s prvim klikom na zemljevidu določili začetno točko potovanja.
- »ciljna točka« - po aktivaciji gumba bomo s prvim klikom na zemljevidu določili ciljno točko potovanja.
- »počisti zemljevid« - zemljevid se postavi v začetno stanje.
- »pomoč« - prikličemo okno z dodatnimi informacijami in navodili.

4.3.2. Implementacija primera »RBFS na zemljevidu«

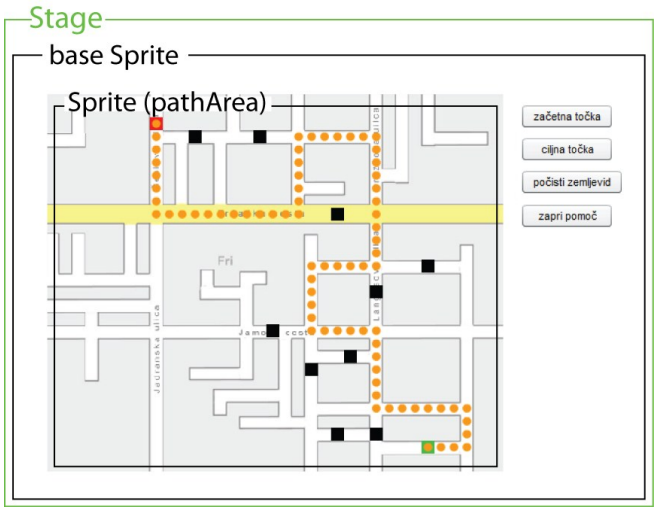
Primer »*RBFS na zemljevidu«* je izvožen v datoteko *RBFSOnMap.swf*. Programska arhitektura je predstavljena s pomočjo diagrama razredov na sliki 36.



Slika 36: Arhitektura razredov za primer »RBFS na zemljevidu«.

Ko se datoteka *RBFSOnMap.swf* odpre v predvajalniku, se pokliče kostruktor glavnega razreda *RBFSOnMap.* Na prizorišču se ustvari osnovni vsebovalnik tipa *Sprite*, kot je ilustrirano na sliki 37. Na osnovni vsebovalnik uvozimo sliko zemljevida kot object tipa *Bitmap*. Dodamo še gumbe tipa *SimpleButton*, ki jim določimo lastnosti, registriramo poslušalce dogodkov ter implementiramo metode za odzivanje na dogodke.

V razredu *ResultView* ustvarimo nov vsebovalnik *pathArea*, kamor izrisujemo rezultat algoritma RBFS in sicer z oranžnimi krogi, ki jih izriše metoda *drawCircle()*. Vsebovalnik *pathArea* vstavimo nad zemljevid, saj hočemo izrisano pot prikazati na zemljevidu.



Slika 37: Sestava grafičnega uporabniškega vmesnika za primer »RBFS na zemljevidu«.

Razred *MapMatrix* ustvari matriko točk na zemljevidu, kamor shranimo posamezno točko kot objekt tipa *MapNode* z atributi (prehodnost, začetna točka, ciljna točka, pozicija na zemljevidu, hevrisitčne ocene).

Podatke za izris poti, določanje naslednikov vozlišč in izračun funkcije *f(n)* vozlišča implementiramo v razredu *RBFS*, ki je implementiran na popolnoma enak način kot v primeru »*RBFS gradi drevo*«, opisanem v poglavju 4.2.2.

5. Zaključek

V diplomski nalogi smo razvili spletno aplikacijo, ki služi kot dodatna pomoč pri razumevanju delovanja algoritma RBFS. Delovanje algoritma je tako mogoče preizkusiti na praktičnih primerih. Animacije so razvite na enostavnih primerih, da se z algoritmom spoznamo in ga lažje razumemo. RBFS postavimo tudi v realno okolje, da uporabnika privabimo k nadaljnjemu raziskovanju in uporabi preiskovalnih algoritmov.
Za izvedbo spletne aplikacije smo izbrali Adobe Flash platformo, ki je primerna za razvoj animacij in vizualnih vsebin. Z razvojnim orodjem Flash lahko kreativno razvijamo grafične objekte in jih vključimo v animacijo. Tako lahko enostavneje prenesemo svoje zamisli v animacije. Programski jezik ActionScript 3.0, z njegovimi knjižnicami, smo uporabili za implementacijo razredov in manipulacijo z objekti. Objektno usmerjeni ActionScript nam je omogočal fleksibilno programsko arhitekturo. Če hočemo na obstoječih primerih preizkusiti druge preiskovalne algoritme, lahko nadomestimo le razred RBFS (implementacija psevdokode vrne rezultat preiskovanja) z željenim algoritmom. Obstoječa implementacija razredov je pregledna in berljiva. Največja težava Flash razvojnih orodij je v tem, da gre za licenčno tehnologijo, ki je dostopna le ob plačilu in morebitni nadaljnji razvoj aplikacije ni odprt.
Pri izdelavi diplomske naloge je največ težav povzročalo začetno nepoznavanje algoritma RBFS. Izbira algoritma, ki ga ne poznamo, je bila namerna, saj smo med učenjem iz obstoječe literature (učbeniki in internet) sprevideli, na kakšen način bi si želeli predstavitev učnih vsebin. Te ideje smo poskušali v čim večji meri prenesti na spletno aplikacijo.

Možnosti za nadaljnje delo in izboljšave je veliko. Obsoječe animacije prikazujejo delovanje algoritma na enostavnih primerih, s katerih se hitro učimo. Aplikacijo bi lahko vsebinsko dopolnili z več primeri uporabe, npr. zelo privlačni so primeri uporabe preiskovalnih algoritmov v enostavnih zabavnih igrah. Uporabnika bi tako vzpodbudili, da se sam preizkusi v izdelavi podobne igre in praktično vključi preiskovalne algoritme. Zanimiva bi bila primerjava z ostalimi preiskovalnimi algoritmi, predvsem A* in IDA*, da uporabnik dobi predvsem časovno in prostorsko predstavo o obnašanju preiskovalnih algoritmov. Zelo zanimivo bi bilo tekmovanje med algoritmi, kjer bi istočasno pognali animacijo preiskovanja dveh ali več algoritmov in spremljali njihovo obnašanje.

Slike

Grafi

Literatura

[1] M. Robnik-Šikonja, Effective use of memory in linear space best first search, FRI, Technical Report, 1996.
[2] I. Bratko. Prolog Programming for Artificial Intelligence. Pearson Addison-Wesley, Harlow, England, 3. izdaja, 2000.
[3] M. Poženel. Inteligentno razpletanje sej pri izgradnji spletnega podatkovnega skladišča, Doktorska disertacija, FRI, Ljubljana, 2010, pogl. 4.
[4] R. Tariq Butt, *Performance Comparison of AI Algorithms,* Magistersko delo, Blekinge Institute of Technology, Ronneby, August, 2008, pogl. 3, 4.
[5] I. Kononenko, M. Robnik-Šikonja. Inteligentni sistemi. FRI, Ljubljana, Slovenija, 1. izdaja, 2010.
[6] (2010) Flash Player penetration. Dostopno na http://www.macromag.com/products/player_census/flashplayer/
[7] (2010) Methodology for Adobe plug-in technology study. Dostopno na http://www.macromag.com/products/player_census/methodology/
[8] (2010)Adobe Flash Platform. Dostopno na http://www.macromag.com/mena/flashplatform/
[9] (2010) SWF. Dostopno na http://en.wikipedia.org/wiki/SWF
[10] (2010)Adobe Flash. Dostopno na http://en.wikipedia.org/wiki/Adobe_Flash
[11] (2010) Flash Professional features. Dostopno na http://www.macromag.com/mena/products/flash/features/
[12] (2010) What customers and analysts are saying. Dostopno na http://www.macromag.com/mena/flashplatform/customers/#analyst_reports
[13] (2010) Adobe Success Story, The New York Times. Dostopno na http://www.macromag.com/cfusion/showcase/index.cfm?event=casestudydetail& casestudyid =703852&loc=en_us
[14] (2010) Flash Platform and NASDAQ. Dostopno na http://www.macromag.com/mena/flashplatform/apps/nasdaq/
[15] (2010) NATO Adopts Adobe Flex for Mission Support System. Dostopno na http://www.macromag.com/aboutadobe/pressroom/pressreleases/200712/121107adobenato.html
[16] (2010) Adobe Success Story, Sony Ericsson Mobile Communications. Dostopno na http://www.macromag.com/cfusion/showcase/index.cfm?event=casestudydetail&casestudyid=739119&loc=en_us
[17] (2010) Adobe Success Story, Disney Publishing Worldwide. Dostopno na http://www.macromag.com/cfusion/showcase/index.cfm?event=casestudydetail&casestudyid=907882&loc=en_us

[18] (2010) Flash Platform and Playfish. Dostopno na
http://www.macromag.com/mena/flashplatform/apps/playfish/
[19] (2010) Flash Platform and DIRECTV. Dostopno na
http://www.macromag.com/mena/flashplatform/apps/directv/
[20] (2010) Flash Platform and SAP BusinessObjects Xcelsius. Dostopno na
http://www.macromag.com/mena/flashplatform/apps/sap/
[21] (2010) Silverlight Vs. Flash Penetration. Dostopno na
http://www.ehow.com/about_6561086_silverlight-vs_-flash-penetration.html
[22] (2010) Real world stats of RIA plugin deployments.Dostopno na
http://www.riastats.com/#
[23] (2010) Adobe Flash Builder. Dostopno na
http://en.wikipedia.org/wiki/Adobe_Flash_Builder
[24] (2010) Adobe Flash Builder 4. Dostopno na http://www.adobe.com/products/flashbuilder/
[25] (2010) Java Code Samples and Apps, Applets. Dostopno na http://java.sun.com/applets/
[26] (2010) Dynamic HTML. Dostopno na http://en.wikipedia.org/wiki/Dynamic_HTML
[27] (2010) Better Experiences for End-Users and Developers with JavaFX. Dostopno na
http://javafx.com/
[28] (2010) HTML5. Dostopno na http://en.wikipedia.org/wiki/HTML5
[29] (2010) Možganom prijazno učenje. Dostopno na
http://www.vodja.net/index.php?blog=1&title=mo-ganom-prijazno-u-
enje&more=1&c=1&tb=1&pb=1
[30] (2010) DataStructures. Dostopno na http://code.google.com/p/polygonal/wiki/DataStructures
[31] (2010) Flash, HTML5 comparison finds neither has performance advantage. Dostopno na
http://www.appleinsider.com/articles/10/03/10/flash_html_5_comparison_finds_neither_has_perf
ormance_advantage.html
[32] (2010) Flash vs. HTML5: What the Platform War Means for You. Dostopno na
http://www.monarchmedia.com/enewsletter_2010-2/flash.html