

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5993

**RAČUNALNA IGRA ZASNOVANA NA
POTRAZI ZA HAMILTONOVSKIM CIKLUSOM
U NEUSMJERENOM GRAFU**

Toma Petrač

Zagreb, lipanj 2019.

Zahvaljujem svome mentoru, prof.dr.sc. Mariju Osvinu Pavčeviću na svakoj podršci, savjetu i ideji čime mi je pomogao pri izradi ovog rada, te što je uvijek bio strpljiv i pristupačan za moje upite.

SADRŽAJ

| | |
|-----------------------------------------------------------------------------------|----|
| SADRŽAJ..... | 3 |
| 1. Uvod..... | 4 |
| 1.1 Hamiltonovost – matematički opis problema | 4 |
| 1.2 <i>Lewis in a Maze</i> | 6 |
| 2. Grafičko sučelje..... | 7 |
| 2.1 Glavni izbornik | 7 |
| 2.2 Stranica s uputama | 8 |
| 2.3 Stranica s najboljim rezultatima | 8 |
| 2.4 Stranica s odabirom razina..... | 9 |
| 2.5 Stranica za pripremu | 10 |
| 2.6 Stranica u igri..... | 11 |
| 2.7 Stranica za pohranu rezultata | 11 |
| 3. Igra | 13 |
| 3.1 Posjećivanje soba..... | 13 |
| 3.2 Gumbi za interakciju | 15 |
| 3.3 Bodovanje i razine | 18 |
| 4. Implementacija | 20 |
| 4.1 Implementacija grafičkog sučelja | 20 |
| 4.2 Implementacija tablice s najboljim rezultatima | 23 |
| 4.3 <i>Netgraph</i> | 25 |
| 4.4 <i>PlayableGraph</i> | 27 |
| 4.5 Povezanost <i>PlayableGraph</i> objekta sa sučeljem i <i>MainApp</i> -om..... | 30 |
| 5. Zaključak..... | 33 |
| 6. Literatura | 35 |

1. Uvod

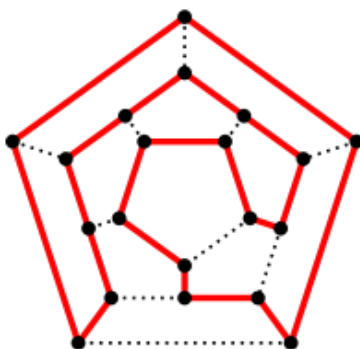
Temeljni principi ovog projekta modelirani su teorijom grafova.

1.1 Hamiltonovost – matematički opis problema

U teoriji grafova, problem hamiltonovosti ispituje postoji li Hamiltonov put ili Hamiltonov ciklus u danom grafu. Hamiltonov put je put koja posjećuje svaki vrh grafa točno jednom. Hamiltonov ciklus je ciklus u grafu (put koji počinje i završava istim vrhom) koji posjećuje svaki vrh osim početno odabranog, točno jednom.

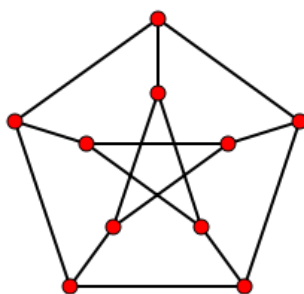
Problem pronalaska Hamiltonova puta je, kao i problem pronalaska Hamiltonova ciklusa u grafu, *NP-Complete*, što znači da je vremenska složenost za te probleme nedeterministički polinomijalna. Nadalje, takve probleme može se riješiti ograničenom klasom algoritama iscrpne pretrage, čija je složenost faktoriijalna.

Graf je hamiltonovski ako sadrži barem jedan Hamiltonov ciklus. Slika 1 prikazuje jedan takav graf.



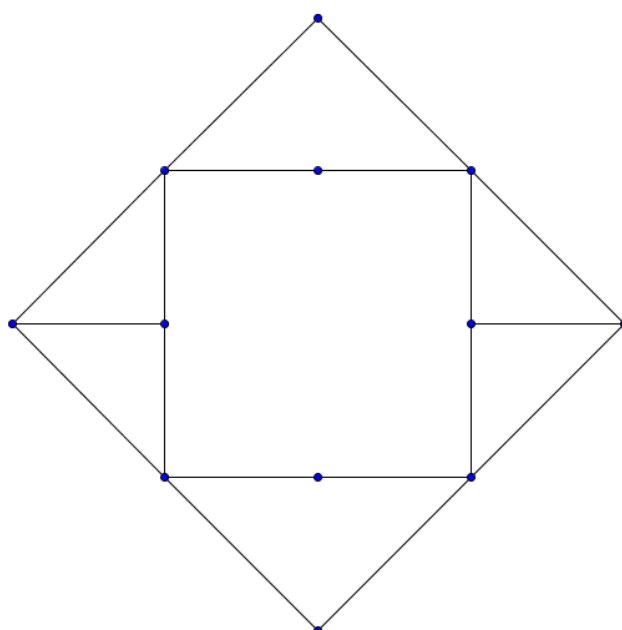
Slika 1. Hamiltonovski graf

Kažemo da je graf skoro hamiltonovski ako sadrži barem jedan Hamiltonov put, ali ne i Hamiltonov ciklus. Slika 2 prikazuje Petersenov graf koji je upravo takav.



Slika 2. Skoro hamiltonovski graf

Slika 3 prikazuje graf koji nije ni skoro hamiltonovski, tj. takav graf u kojem ne postoji nijedan hamiltonovski put.



Slika 3. Nehamiltonovski graf

Ne postoji jednostavan nužan i dovoljan uvjet za provjeru je li graf hamiltonovski. Jedni od najpoznatijih dovoljnih uvjeta iskazani su teoremima Diraca i Orea:

- **Dirac** (1952.) – jednostavan graf s n vrhova ($n \geq 3$) je hamiltonovski ako je stupanj svakog njegovog vrha $n/2$ ili veći.
- **Ore** (1960.) – jednostavan graf s n vrhova ($n \geq 3$) je hamiltonovski ako je suma stupnjeva za svaki par ne-susjednih vrhova n ili veća.

Prisjetimo se, stupanj vrha grafa je ukupan broj bridova s kojima je taj vrh incidentan.

1.2 *Lewis in a Maze*

Ideja same igre je vrlo jednostavna te je usko povezana s matematičkim definicijama navedenima u prethodnom odjeljku. Naime, pred igračem se pojavljuju grafovi za koje je potrebno pronaći Hamiltonove putove posjećivanjem, tj. klikanjem vrhova onim redoslijedom kojim prolazi barem jedan Hamiltonov put zadanog grafa.

Uz pretpostavku da mnogi igrači nisu upoznati s matematikom problema, navedenom u odjeljku 1.1, upute igre opisane su na jednostavniji način, te je osmišljena priča za igru.

Radi se o tome da se dječak imenom Lewis (ime poznatog vozača formule 1, Lewisa Hamiltona, što predstavlja referencu na hamiltonovost) nalazi pred neobičnim labirintima. Za svaki labirint pred kojim se nalazi dobije tlocrt. Poanta je da mora posjetiti svaku sobu labirinta jer se u svakoj od njih nalazi novčić, a potrebno ih je sakupiti sve. Međutim, svaki put nakon što izađe iz jedne sobe labirinta, vrata se iza njega zatvore, te se više ne može vratiti. Zato mu igrač mora pomoći isplanirati rutu prije, tako da mu pokaže točno kojim redoslijedom mora posjećivati sobe da bi ih uspio sve posjetiti i skupiti sve novčiće u labirintu. Također, neki labirinti nisu rješivi jer ne postoji ruta kojom bi Lewis mogao sakupiti sve novčiće. Za takve labirinte potrebno je javiti Lewisu da ni ne ulazi jer će inače ostati zatočen.

Labirinti su modelirani grafovima, tako da svaka soba labirinta predstavlja jedan vrh grafa, a svaki hodnik predstavlja jedan brid. Igrač zapravo pokušava pronaći Hamiltonove putove u slučajno zadanim grafovima. Nerješivost labirinta analogna je nepostojanju Hamiltonova puta u grafu.

2. Grafičko sučelje

Kao i u svakoj računalnoj igri (ili bi barem trebalo biti), grafičko sučelje u igri *Lewis in a Maze* je intuitivno i jednostavno, prije svega zato što su imena gumbi samoobjašnjavajuća, a sve opcije lako vidljive.

2.1 Glavni izbornik

Pokretanjem aplikacije korisnik će odmah moći vidjeti glavni izbornik, čiji je *Frame* prilično jednostavan te sadrži jednu labelu i četiri gumba. Labela se nalazi na vrhu, te je pisana jako velikim fontom jer sadrži ime igre. Gumbi su redom (odozgo prema dolje):

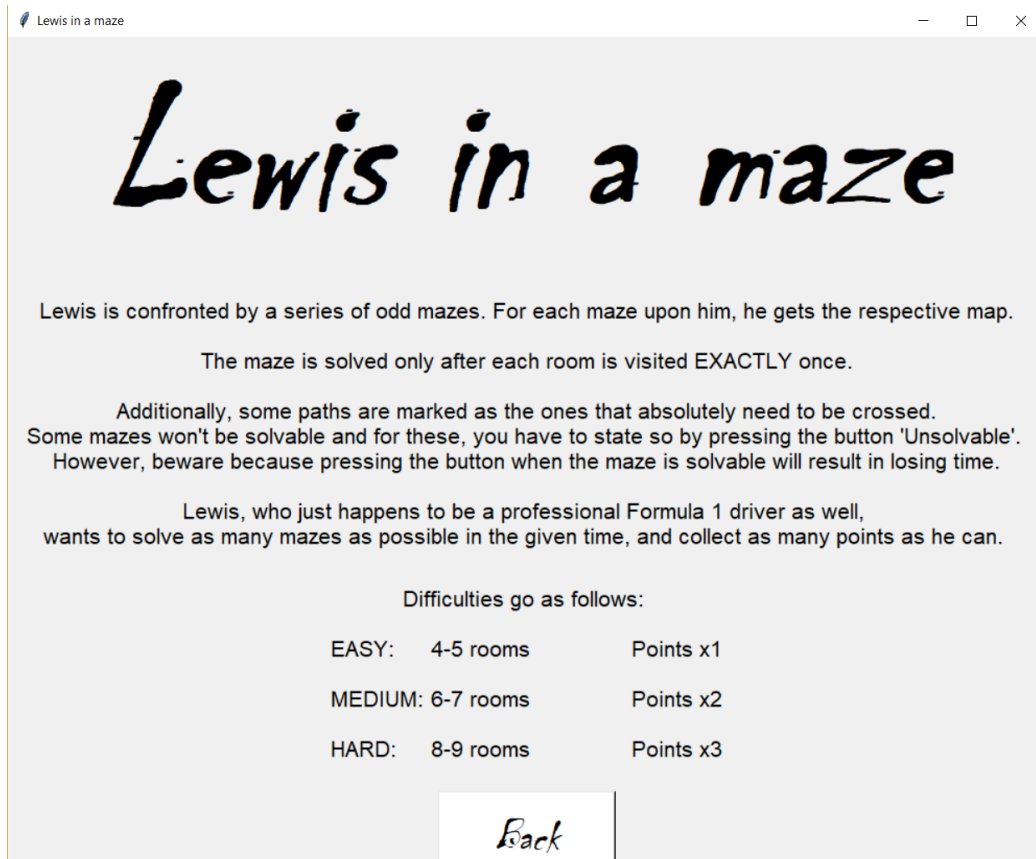
- *Play* (Igraj): Otvara novu stranicu (*Frame*) grafičkog sučelja igre u kojem se odabire težina.
- *Highscores* (Najbolji rezultati): Otvara novu stranicu (*Frame*) grafičkog sučelja u kojem se može vidjeti tablica najboljih rezultata.
- *Instructions* (Upute): Otvara novu stranicu (*Frame*) grafičkog sučelja na kojoj su navedene upute kako igrati.
- *Quit* (Izlaz): Izlaz iz igre.



Slika 4. Glavni izbornik

2.2 Stranica s uputama

Stranica s uputama prilično je jednostavna. Osim glavne, velike labele s imenom igre, sadrži i labele s opisima i uputama za igru te gumb za povratak na glavni izbornik.

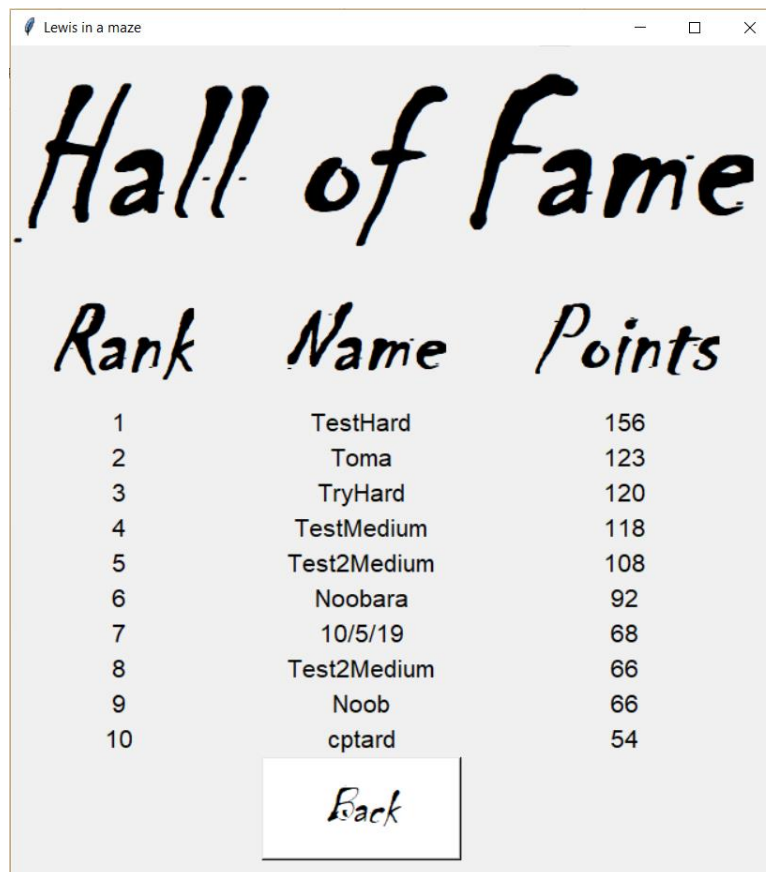


Slika 5. Stranica s uputama

2.3 Stranica s najboljim rezultatima

Stranica s najboljim rezultatima također se sastoji od nekoliko labela i jednog gumba za povratak na glavni izbornik. Uvijek se prikazuje deset najboljih rezultata u formatu:

Rang | Ime | Bodovi



| Rank | Name | Points |
|------|-------------|--------|
| 1 | TestHard | 156 |
| 2 | Toma | 123 |
| 3 | TryHard | 120 |
| 4 | TestMedium | 118 |
| 5 | Test2Medium | 108 |
| 6 | Noobara | 92 |
| 7 | 10/5/19 | 68 |
| 8 | Test2Medium | 66 |
| 9 | Noob | 66 |
| 10 | cptard | 54 |

Back

Slika 6. Stranica s najboljim rezultatima

2.4 Stranica s odabirom razina

Ovoj stranici pristupa se pritiskom gumba *Play* na glavnom izborniku. Stranica je također vrlo jednostavna te sadrži jednu veliku labelu (za prikaz odabira razine), te četiri gumba, od kojih su tri za odabir razine, te jedan za povratak na glavni izbornik. O razlici između razina govorit će se u poglavlju 3.



Slika 7. Stranica za odabir razine

2.5 Stranica za pripremu

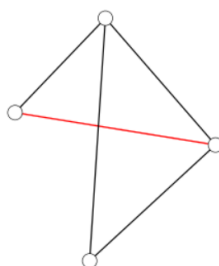
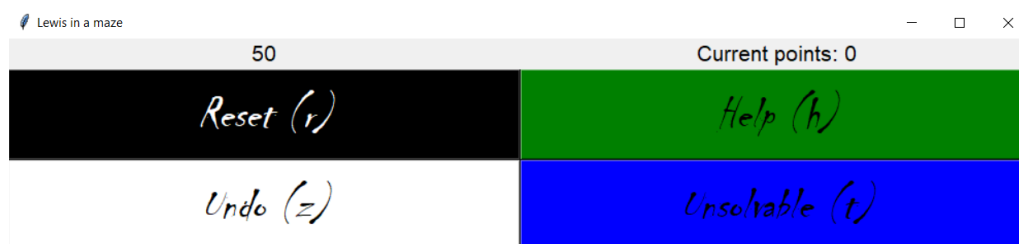
Nakon odabira razine, prikazuje se stranica za pripremu koja sadrži samo dva gumba: gumb za pokretanje igre i gumb za povratak na odabir razine. Razlog zbog kojeg ova stranica postoji je jednostavno priroda igre. Naime, s obzirom da se igra temelji na maksimalnoj iskoristivosti vremena, činilo se intuitivnije za igrača da igra stvarno krene tek nakon pritiska gumba *Start*, a ne *Easy* ili *Hard*.



Slika 8. Stranica za pripremu

2.6 Stranica u igri

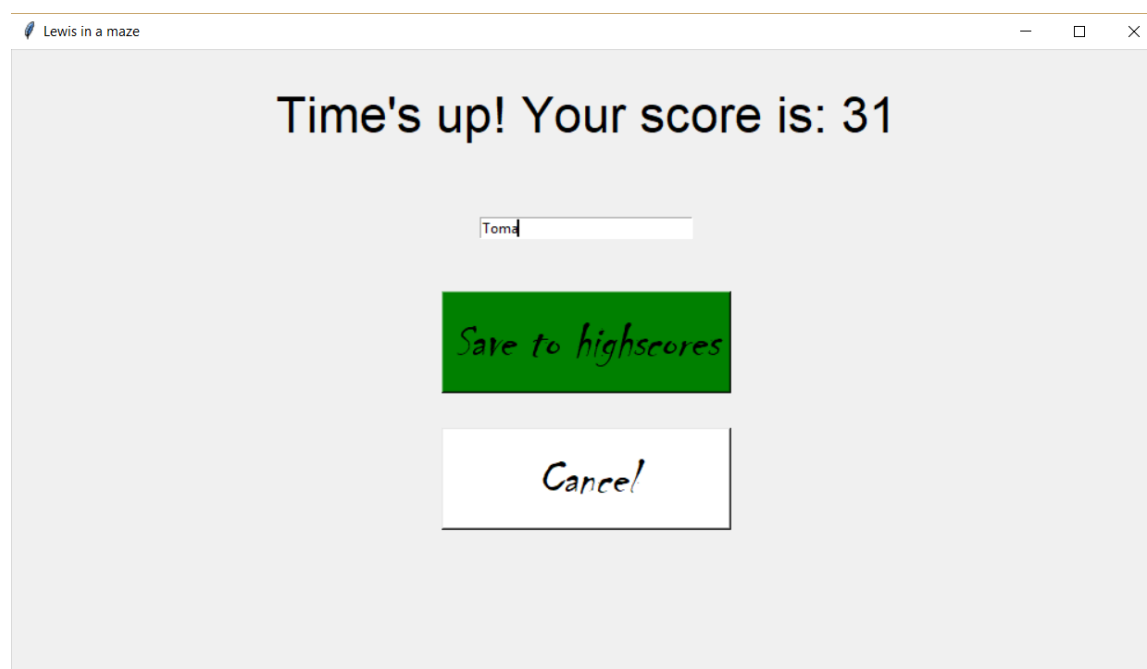
Najvažnija stranica je upravo ona dok igra traje. Stranica sadrži dvije labele na vrhu ekrana za prikaz preostalog vremena i trenutnih bodova. Zatim sadrži četiri gumba, po dva sa svake strane. Njihove funkcije bit će detaljno objašnjene u poglavlju 3. Ispod tih gumba je prostor gdje se crtaju grafovi, odnosno tlocrti mapa labirinta koje igrač treba riješiti.



Slika 9. Stranica u igri

2.7 Stranica za pohranu rezultata

Nakon što je igra gotova, prikazat će se stranica za pohranu rezultata. Na vrhu stranice nalazi se labela koja kaže da je igra gotova te prikazuje ukupan broj bodova koji je igrač postigao. Ispod nje nalazi se polje za upis imena, te dva gumba: jedan za pohranu rezultata, te drugi za poništavanje rezultata (ukoliko igrač ne želi pohraniti rezultat). Pritiskom na oba gumba igrač se vraća na glavni izbornik (osim ako je pokušao spremići rezultat ostavivši polje za unos imena praznim).



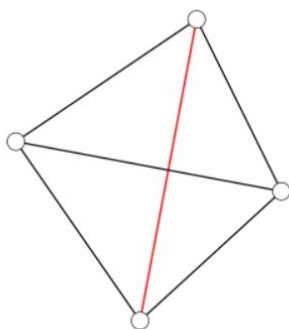
Slika 10. Stranica za pohranu rezultata

3. Igra

Odmah nakon pritiska na gumb *Start*, prikazat će se stranica za igru (kako je pokazano u prethodnom poglavlju), i započet će odbrojavanje. Koncept je prilično jednostavan. Sve što igrač može raditi tijekom igre je označavati sobe labirinta koje hoće posjetiti, i pritisnuti gumbe za interakciju.

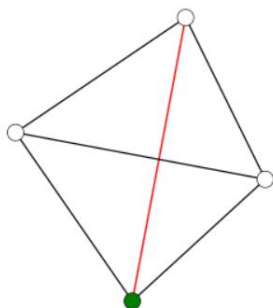
3.1 Posjećivanje soba

Posjećivanje soba je temelj igre jer ono predstavlja glavni način rješavanja labirinta. Sve su sobe na početku slobodne te igrač bira koju sobu će prvu posjetiti. Neposjećene sobe označene su bijelim kružićima.



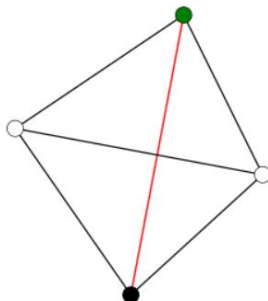
Slika 11. Labirint bez posjećenih soba

Nakon što igrač posjeti sobu, kružić koji je označio pritiskom desnog klika mišem, postaje zelen.



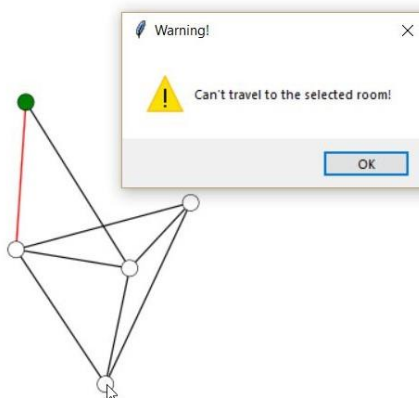
Slika 12. Labirint s jednom posjećenom sobom

Pravilo igre je da, jednom kad igrač posjeti sobu, ne smije je ponovno posjetiti. Zato će sve prethodno posjećene sobe osim one u kojoj se igrač trenutno nalazi (koja je prikazana zelenim kružićem) biti označene crnim kružićem.



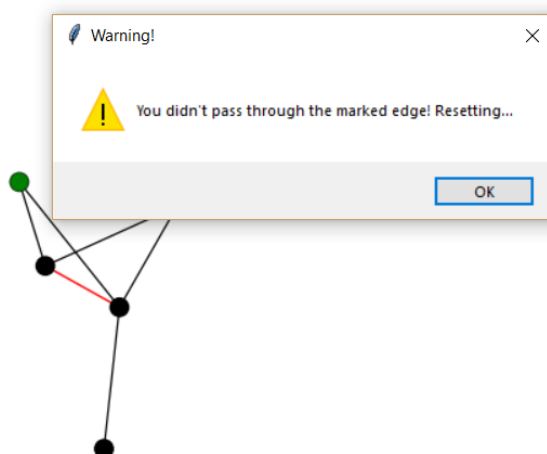
Slika 13. Labirint s dvije posjećene sobe

Postoje slučajevi kada je iz jedne sobe nemoguće prijeći u neku drugu neposjećenu. U tim slučajevima, ukoliko je igrač pokušao posjetiti neposjećenu sobu koju nije mogao posjetiti iz trenutne jer nisu direktno povezane hodnikom, prikazat će se prozor s upozorenjem.



Slika 14. Nepovezane sobe

Ako je igrač posjetio sve sobe, ali nije prošao označenim hodnikom, otvorit će se novi prozor s upozorenjem, a pritiskom gumba „OK“, sve sobe vratit će se u neposjećeno stanje.

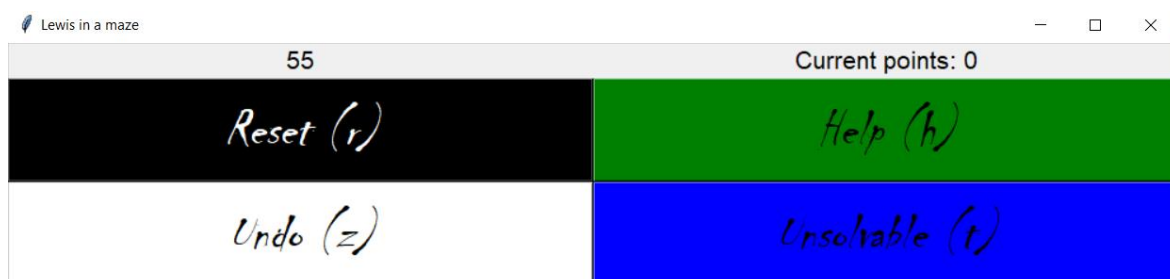


Slika 15. Neprolazak označenim hodnikom

Nakon što su sve sobe posjećene (uz dodatak da je igrač prošao označenim hodnikom), igrač osvaja bodove (vidi odjeljak 3.3) i na ekranu se pojavljuje novi labirint kojeg treba riješiti.

3.2 Gumbi za interakciju

Kao što je već prikazano u prethodnom poglavlju, dok igra traje, igrač može koristiti četiri gumba. Na kraju teksta svakog gumba navedeno je po jedno slovo u zagradi koje označava prečac preko tipkovnice za pripadajući gumb.



Slika 16. Gumbi

Funkcije gumba su sljedeće:

Reset (r)

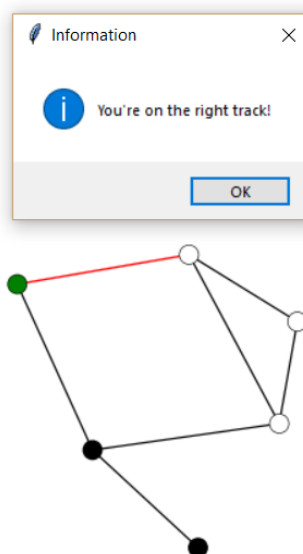
- Resetira trenutni labirint tako da sve sobe vraća u neposjećeno stanje

Undo (z)

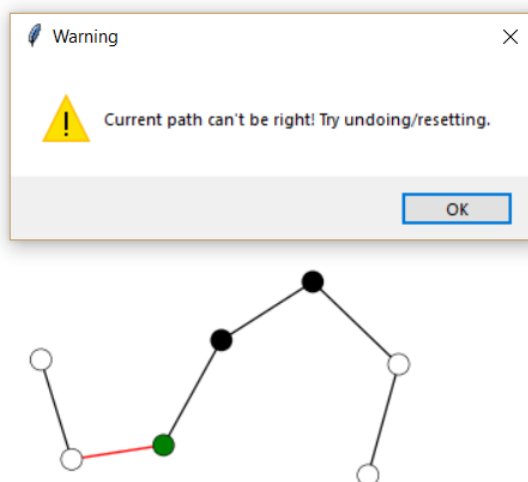
- Poništava zadnji potez igrača na način da se igrač vraća u sobu u kojoj je bio jedan potez prije zadnjeg.

Help (h):

- Igrač u bilo kojem trenutku može pozvati pomoć pritiskom na ovaj gumb.
- Dva su moguća scenarija:
 - Igrač je trenutno na dobrom putu, labirint se može riješiti bez korištenja gumba za resetiranje (*reset*) i poništavanja posljednjeg poteza (*undo*).
 - Igrač nije na dobrom putu jer iz trenutnog stanja posjećenosti svih soba nije moguće riješiti labirint; igrač će morati koristiti gumbe za resetiranje (*reset*) ili poništavanje posljednjeg poteza (*undo*). U krajnjem slučaju, moguće je da je labirint doista nerješiv (vidi sljedeći gumb).
- VAŽNA NAPOMENA: Korištenje ovog gumba rezultira automatskim spuštanjem trenutnog broja bodova na nulu!



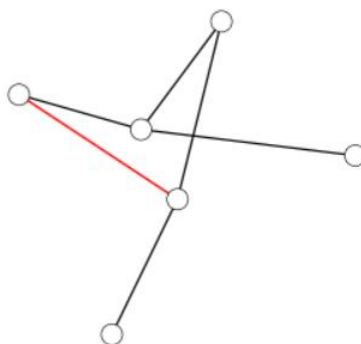
Slika 17. Gumb za pomoć: točan put



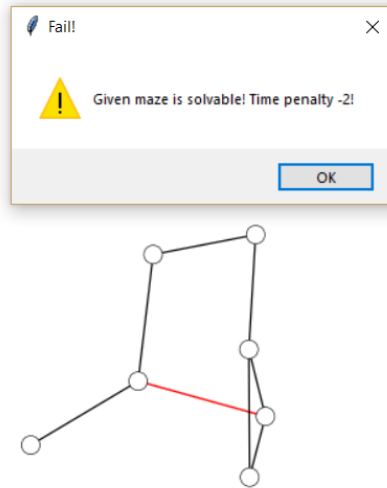
Slika 18. Gumb za pomoć: Krivi put

Unsolvable (t)

- Postoje slučajevi kada labirint nikako nije moguće riješiti jer ne postoji takav put da se svaka soba posjeti točno jednom, uz dodatak da treba proći i označenim hodnikom.
- U tim slučajevima potrebno je pritisnuti ovaj gumb, te, ukoliko je labirint doista nerješiv, igrač će dobiti bodove, i odmah će se prikazati novi labirint.
- Međutim, ukoliko je labirint rješiv, a igrač je pritisnuo ovaj gumb, otvorit će se novi prozor s upozorenjem da je labirint zapravo rješiv, a igraču će se oduzeti dvije sekunde ukupnog vremena koje ima na raspolaganju.



Slika 19. Primjer nerješivog labirinta



Slika 20. Pogrešno korištenje gumba 'Unsolvable'

3.3 Bodovanje i razine

Osnovna formula za bodovanje glasi:

$$Bodovi = 2 \cdot m + \sum_i^n x_i \quad (3.1)$$

m – Ukupan broj nerješivih labirinta za koje je igrač pritisnuo gumb *Unsolvable*.

n – Ukupan broj riješenih rješivih labirinta.

x_i – Ukupan broj soba i -tog labirinta.

Ukratko, za svako ispravno korištenje gumba *Unsolvable*, igrač dobiva 2 boda, a za svaki riješeni labirint dobiva bodova koliko taj labirint sadrži soba.

Tablica 1 prikazuje osobitosti za svaku razinu.

Tablica 1. Razine

| Razina | Broj soba u labirintu | Ukupno vrijeme | Vremenski dodatak | Koeficijent za bodove |
|---------------|-----------------------|----------------|-------------------|-----------------------|
| <i>Easy</i> | 4-5 | 60 sekundi | 0 sekundi | 1 |
| <i>Medium</i> | 6-7 | 60 sekundi | 3 sekunde | 2 |
| <i>Hard</i> | 8-9 | 90 sekundi | 5 sekunde | 3 |

Broj soba u labirintu označava slučajan broj od dvaju navedenih. Na primjer, za razinu *Easy*, broj soba u svakom labirintu bit će 4 ili 5.

Ukupno vrijeme je vrijeme jedne igre. Cilj igre je u tom vremenu riješiti što više labirinta, a samim time skupiti što više bodova.

Vremenski dodatak predstavlja broj sekundi dodanih ukupnom preostalom vremenu za svaki ispravno riješeni labirint. Međutim, to ne vrijedi za nerješive labirinte: Ako je labirint nerješiv te igrač to primijeti i pritisne gumb *Unsolvable*, igrač će dobiti bodove, ali neće biti vremenskog dodatka.

Naposljetku, koeficijent za bodove je broj kojim se na kraju igre množi ukupan broj bodova ostvaren u igri (3.1).

Igra je napravljena tako da potiče igrača na što veći izazov. Da bi ostvario što veći broj bodova, potrebno je odabrati najtežu razinu.

4. Implementacija

Računalna igra *Lewis in a Maze* napisana je u programskom jeziku *Python*, koji zbog svoje jednostavnosti, svestranosti i dinamičnosti slovi kao jedan od najpopularnijih programskih jezika današnjice.

Sam projekt čine sljedeće *.py* datoteke:

- *gui.py* – U toj datoteci definirano je grafičko sučelje. Osim toga, tu se nalaze i mnoge druge funkcionalnosti te povezanost sa vanjskom bibliotekom *netgraph*. Pokretanjem upravo te datoteke pokreće se i sama igra.
- *database.py* – Datoteka u kojoj je kreirana tablica za najbolje rezultate u obliku *.db* datoteke

Osim tih datoteka, ključna je i vanjska biblioteka ***netgraph***, čija je glavna namjena crtanje grafova. U sklopu ovog projekta, ta je biblioteka modificirana (vidi odjeljak 4.3).

U ovom poglavlju razmatrat će se realizacija pojedinih dijelova aplikacije, kao i realizacija korištenih algoritama.

4.1 Implementacija grafičkog sučelja

Osnova realizacije grafičkog sučelja jedna je od standardnih biblioteka *Python*-a: modul ***tkinter***. Ukratko, da bi se ostvarilo grafičko sučelje pomoću tog modula, potrebno je (naravno, uz unos modula, tj. *importanje* biblioteke) inicijalizirati novi *Tk* objekt koji će predstavljati korijen, odnosno glavni prozor. Zatim se na taj prozor mogu slagati tzv. *widgeti* (što su zapravo bilo kakvi gumbi, labele i sl.). Na kraju je još potrebno pozvati metodu *mainloop* nad korijenskim *Tk* objektom. Ta metoda je apsolutno nužna jer čeka na događaje i osvježava grafičko sučelje u beskonačnoj petlji. Slika 21 prikazuje navedenu osnovu napisanu za GUI igre *Lewis in a Maze*.

```
424 app = MainApp()  
425 app.mainloop()
```

Slika 21. Osnova *tkinter* aplikacije

Klasa *MainApp* nasljeđuje klasu *Tk*.

Nadalje, slika 22 prikazuje kôd klase *MainApp*.

```

29 class MainApp(Tk):
30
31     def __init__(self, *args, **kwargs):
32
33         super(MainApp, self).__init__(*args, **kwargs)
34
35         Tk.wm_title(self, "Lewis in a maze")
36
37         container = Frame(self)
38         container.pack(side="top", fill="both", expand=True)
39         container.grid_rowconfigure(0, weight=1)
40         container.grid_columnconfigure(0, weight=1)
41
42         self.frames = {}
43         self.connector = connect("Highscores.db")
44         self.cursor = self.connector.cursor()
45
46         for F in (MainMenuPage, HighscoresPage, InstructionsPage,
47                 SelectDiffPage, CustomGraphPage, GamePage, SaveToHighscoresPage):
48             frame = F(container, self)
49             self.frames[F] = frame
50             frame.grid(row=0, column=0, sticky="nsew")
51
52         self.show_frame(MainMenuPage)
53
54     def show_frame(self, cont):
55         frame = self.frames[cont]
56         frame.tkraise()
57

```

Slika 22. MainApp

Grafičko sučelje realizirano je na principu *container*-a i *frame*-ova. Naime, svaka stranica je zapravo jedan *Frame*, čiji je *container* roditelj, a *MainApp* upravljač. Prilikom pokretanja aplikacije, svaka će se stranica inicijalizirati kao zaseban *Frame*, a naposljetku će se prikazati stranica s glavnim izbornikom (*MainMenuPage*). Iako aplikacija ima inicijalizirane sve *Frame*-ove, može ih prikazivati samo jednog po jednog funkcijom *show_frame*.

Slika 23 prikazuje kôd klase *MainMenuPage*.

```

60 class MainMenuPage(Frame):
61
62     def __init__(self, parent, controller):
63         super(MainMenuPage, self).__init__(parent)
64
65         title_label = Label(self, text="Lewis in a maze", font=("Chiller", 128, "bold", "italic"))
66         title_label.pack(pady=15, padx=15)
67
68         button_play = Button(self, text="Play", font=BUTTON_FONT, width=10, bg="green",
69                             cursor=CURSOR, command=lambda: controller.show_frame(SelectDiffPage))
70         button_play.pack(padx=15, pady=15)
71
72         button_highs = Button(self, text="Highscores", font=BUTTON_FONT, width=10, bg="black", fg="white",
73                               cursor=CURSOR, command=lambda: controller.show_frame(HighscoresPage))
74         button_highs.pack(padx=15, pady=15)
75
76         button_instr = Button(self, text="Instructions", font=BUTTON_FONT, width=10, bg="cyan",
77                               cursor=CURSOR, command=lambda: controller.show_frame(InstructionsPage))
78         button_instr.pack(padx=15, pady=15)
79
80         button_quit = Button(self, text="Quit", font=BUTTON_FONT, width=10, bg="white",
81                              cursor=CURSOR, command=parent.quit)
82         button_quit.pack(padx=15, pady=15)

```

Slika 23. *MainMenuPage*

Može se primijetiti da su argumenti za inicijalizaciju objekta *MainMenuPage*, osim *self*-a, upravo roditelj i upravljač, kao što je navedeno ranije. Ako se vratimo na sliku 22, primjećujemo da se svaka stranica inicijalizira s dodatnim argumentima: *container* (što će biti roditelj) i *self*, što u *MainApp*-u predstavlja upravo taj objekt *MainApp*-a kao upravljač za navedenu stranicu.

BUTTON_FONT i *CURSOR* su konstante za kursor i font gumba, definirane na početku kôda.

Dodatni argument *command* za svaki gumb realiziran je *lambda* izrazima. To je opće poznata i prihvaćena praksa za ovakvo korištenje *tkinter* modula, a predstavlja jedan od načina prosljeđivanja funkcija s argumentima.

Ostale stranice ostvarene su na sličan način. Međutim, stranica za igru sadrži neke dodatke, jer osim klasičnih gumba i labela, na toj stranici se crtaju i grafovi (labirinti). Da bi se to postiglo, koristi se tzv. *canvas* (platno) i biblioteka *matplotlib*, odnosno *pyplot*. Slika 24 prikazuje dio kôda stranice za igru.

```

236 self.fig, ax = plt.subplots(1, 1)
237 ax.set(xlim=[-2, 2], ylim=[-2, 2])
238
239 self.canvas = FigureCanvasTkAgg(self.fig, master=self)

```

Slika 24. *Canvas* (platno)

Plt je ovdje skraćenica za uvezeni modul *matplotlib.pyplot*. Funkcija *subplots* prima broj stupaca i redaka u novom *Figure* objektu, a vraća taj *Figure* objekt i *Axes* objekt. *Figure* predstavlja nešto slično prozoru po kojem se crtaju objekti a *Axes* predstavlja x i y osi koordinatnog sustava *Figure*-a.

FigureCanvasTkAgg predstavlja objekt koji povezuje *Figure* iz biblioteke *matplotlib* s grafičkim sučeljem *tkinter*-a.

Slika 25 prikazuje uporabu i korištene metode *canvas*-a.

```
249 self.canvas.draw()
250 self.canvas.get_tk_widget().grid(column=0, row=3, columnspan=2, sticky="nsew")
```

Slika 25. Metode *canvas*-a

Kao što možemo vidjeti u liniji 250, metoda *get_tk_widget* znatno nam olakšava korištenje *canvas*-a jer ga sada možemo tretirati kao bilo koji *widget* iz *tkinter* sučelja. Ako se prisjetimo kako izgleda stranica u igri (poglavlje 2.6), znamo da sadrži *grid* od četiriju redaka i dvaju stupaca. S obzirom da indeksiranje u *Python*-u uvijek kreće od 0, redak naveden u funkciji *grid* bit će 3. *Columnspan=2* znači da će se prostirati kroz oba stupca, a *sticky="nsew"*, znači da će se širiti na sve strane, do svih krajeva prozora.

4.2 Implementacija tablice s najboljim rezultatima

Tablica s najboljim rezultatima izvedena je kao *.db* (*database*) datoteka. To je zapravo baza podataka čiji su ključevi ime i bodovi, a primarni ključ je indeks koji se automatski inkrementira svakim novim unosom. Jedna od biblioteka koju *Python* koristi za upravljanje bazama podataka je *sqlite3*.

Slika 26 prikazuje kôd kojim se stvara ta tablica (što je ujedno i sadržaj datoteke *databases.py*)

```

1      import sqlite3
2
3      con = sqlite3.connect("Highscores.db")
4      cursor = con.cursor()
5      queryTable = "CREATE TABLE Highscores (" \
6                  "ID INTEGER PRIMARY KEY AUTOINCREMENT, " \
7                  "Ime TEXT, " \
8                  "Points INTEGER )"
9
10     cursor.execute(queryTable)
11     con.commit()
12     con.close()

```

Slika 26. Databases.py

Da bismo mogli izvesti neki SQL upit, potrebno je najprije spojiti se na željenu bazu metodom *connect*. Zatim nad objektom koji nam je ta naredba vratila (konektorom) treba pozvati *cursor*, jer se tek nad njime može pozvati metoda *execute* koja izvršava SQL upite. Naposljetku, nad konektorom je potrebno pozvati metodu *commit* koja će zapamtiti i pohraniti promjene nad bazom, a zatim je potrebno zatvoriti vezu s bazom (*close*) ako se više neće koristiti.

Realizacija unosa u bazu podataka prikazana je na slici 27.

```

392     def insert_to_highscores(self):
393         name = self.entry.get()
394         if name:
395             insert = "INSERT INTO Highscores (Ime, Points) VALUES ('{0}',{1})".format(name, self.points)
396             self.controller.cursor.execute(insert)
397             self.controller.connector.commit()

```

Slika 27. Unos u bazu podataka

Prije unosa potrebno je provjeriti je li igrač upisao ime u *entry*. Sam unos ostvaren je prilično jednostavnim SQL upitom. Prisjetimo se, *controller* je zapravo *MainApp* koji sadrži članske varijable konektora na bazu podataka i njenog kursora.

Slika 28 prikazuje upit za selekciju od deset rezultata poredanih po broju bodova.

```

115     def update_scores(self):
116         query = "SELECT * FROM Highscores ORDER BY Points DESC LIMIT 10"
117         table = self.controller.cursor.execute(query)

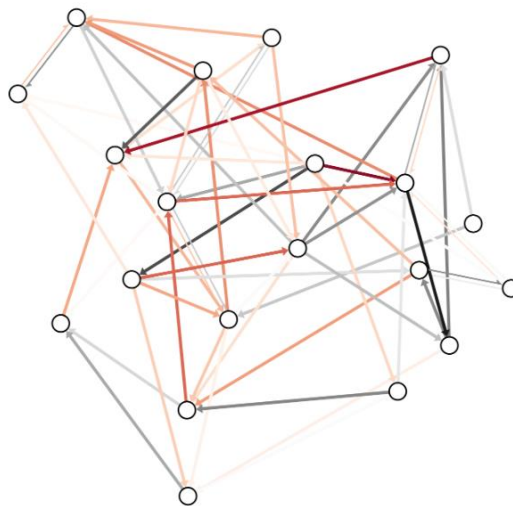
```

Slika 28. Selekcija najboljih rezultata

4.3 Netgraph

Netgraph je vanjska *Python* biblioteka za kvalitetno iscrtavanje grafova. Moguće je crtanje težinskih, bestežinskih, usmjerenih ili neusmjerenih grafova različitih boja. Biblioteka koristi poznate module *numpy* i *matplotlib*, te je tzv. *open-source* projekt na *github* repozitoriju.

Naglasak *netgraph* modula je na interaktivnosti. S obzirom da su algoritmi za crtanje slučajnih grafova prilično složeni (ako bismo htjeli da „lijepo“ izgledaju), *netgraph* daje mogućnost da se, nakon što je graf nacrtan, pozicije vrhova pomiču mišem. Iako ta mogućnost u ovom radu nije iskorištena, neka svojstva takvih interaktivnih grafova ipak jesu.



Slika 29. Graf nacrtan pomoću *Netgraph-a*

U ovom odjeljku neće se opisivati detaljno sve mogućnosti ove biblioteke jer ih ima jako puno, a mnogo ih i nije iskorišteno u radu.

Ono što je najbitnije za ovaj rad je klasa *Graph*. Tu klasu nasljeđuje mnogo drugih klasa s kojima se može raditi, a nasljeđuje je i ona napisana za igru: *PlayableGraph* (vidi odjeljak 4.4). Za inicijalizaciju *Graph* objekta, minimalno je potrebno proslijediti argument *graph*. Taj argument može biti raznih formata:

- Lista bridova – *iterable* objekt parova (*izvor*, *odredište*)
- $N \times N$ matrica susjedstva, gdje je N broj vrhova grafova. Odsutnost veze označena je nulom.

- *igraph.Graph* objekt
- *networkx.Graph* objekt

igraph i *networkx* također su vanjske biblioteke za upravljanje grafovima u Pythonu.

```
1524 class Graph(object):
1525
1526     def __init__(self, graph, node_positions=None, node_labels=None, edge_labels=None, edge_cmap='RdGy', ax=None, **kwargs):
1527
1528         self.draw(graph, node_positions, node_labels, edge_labels, edge_cmap, ax, **kwargs)
```

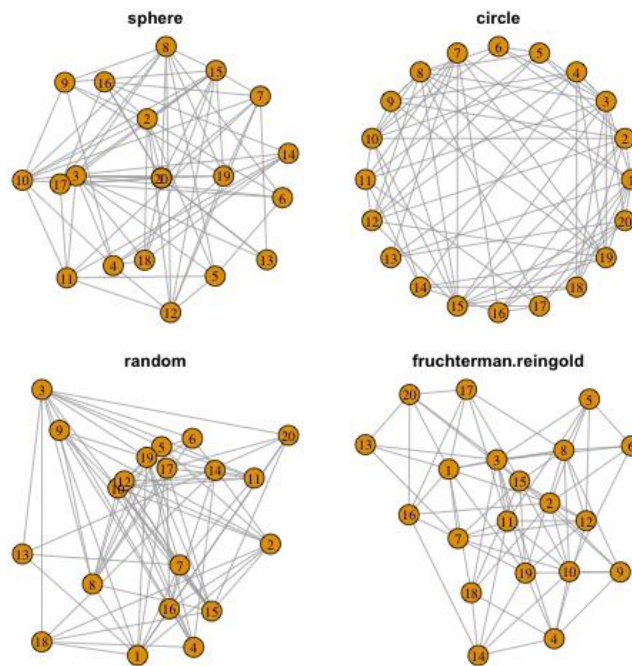
Slika 30. Klasa *Graph*

Inicijalizacijom jednog *Graph* objekta odmah se poziva funkcija za crtanje. Ostali mogući argumenti su:

- *node_positions*: rječnik kojemu je ključ vrh grafa, a vrijednost par (x, y) . Ukoliko je *graph* matrica susjedstva, vrhovi moraju biti cijeli brojevi u rasponu n , gdje je n ukupan broj vrhova grafa.
- *node_labels*: rječnik kojemu je ključ vrh grafa, a vrijednost *String*.
- *edge_labels*: rječnik kojemu je ključ par (*izvor, odredište*), a vrijednost *String*.
- *ax*: instanca *matplotlib.axis*, tj. objekta na kojem se crta. Ako je *None*, instancirat će se novi funkcijom *plt.gca()*.

Naposljetku, ono što je još važno spomenuti je način na koji se ostvaruju položaji vrhova grafa. Koristi se Fruchterman-Reingoldov raspored koji se ostvaruje tzv. *force-directed* (silom usmjerenim) algoritmima. Cilj takvog rasporeda je pozicioniranje vrhova grafa tako da su svi bridovi približno jednake duljine, uz što manje preklapanja bridova.

Slika 31 prikazuje četiri različita rasporeda vrhova grafa. Iako grafovi sadrže mnogo vrhova, čini se da je za potrebe ovog rada ipak najuvjerljiviji upravo Fruchterman-Reingoldov jer se najbolje vidi koji je vrh povezan s kojim, a i razmaci između vrhova su u prosjeku veći.



Slika 31. Različiti rasporedi vrhova grafa

4.4 PlayableGraph

PlayableGraph nadodana je klasa u *netgraph* modulu, sa svrhom enkapsuliranja jednog labirinta, tj. grafa za koji je potrebno pronaći Hamiltonov put. Klasa nasljeđuje klasu *Graph* opisanu u prethodnom odjeljku.

Klasa sadrži mnogo članskih varijabli, a slika 32 prikazuje neke od njih.

```

2185 class PlayableGraph(Graph):
2186
2187     def __init__(self, *args, **kwargs):
2188         Graph.__init__(self, *args, **kwargs)
2189
2190         self.marked_edge = list(self.edge_artists.keys())[0]
2191         self.edge_artists[self.marked_edge].set_color('red')
2192
2193         self.no_vertices = len(self.node_positions)
2194         self.adjacency_matrix = [[] for i in range(self.no_vertices)]
2195
2196         self.visited = []
2197
2198         for edge in self.edge_list:
2199             src = edge[0]
2200             dest = edge[1]
2201
2202             self.adjacency_matrix[src].append(dest)
2203             self.adjacency_matrix[dest].append(src)
2204
2205         self.hamiltonian_paths = []

```

Slika 32. *PlayableGraph* - članske varijable

Klasa *Graph* između ostalog sadrži dva rječnika. To su *node_artists* čiji su ključevi vrhovi grafa, a vrijednosti krugovi iz *matplotlib.patches* modula, te *edge_artists* čiji su ključevi bridovi, tj. parovi (*izvor, odredište*), a vrijednosti linije, također iz *matplotlib*-a. Članske varijable na slici 32 redom su:

- *marked_edge* – označeni brid, tj. onaj brid kojim će biti potrebno proći da bi graf bio riješen. S obzirom da je svejedno koji je to brid, ovdje se uvijek indeksira 0-ti brid (ionako su pozicije bridova slučajne). Linija 2191 mijenja boju tog brida u crvenu.
- *no_vertices* – ukupan broj vrhova grafa.
- *adjacency_matrix* – matrica susjedstva grafa.
- *visited* – lista trenutno posjećenih vrhova grafa. Vrhovi su označeni kao cijeli brojevi iz intervala $[0, n - 1]$, gdje je n ukupan broj vrhova.
- *hamiltonian_paths* – lista svih Hamiltonovih putova grafa.

Pritiskom na vrh grafa (zapravo pritiskom na bilo koju poziciju podloge) program će ispitati mnogo uvjeta. Na slici 33 prikazan je kôd koji se izvodi prilikom klika mišem na novi, neposjećeni vrh grafa.

```

else:
    # Clicked on new artist
    previous_node = None
    if len(self._selected_artists) > 0:
        previous_node = self._get_node_by_artist(self._selected_artists[len(self._selected_artists) - 1])
        self._selected_artists[len(self._selected_artists) - 1].set_color('black')
    next_node = self._get_node_by_artist(artist)
    if not self.areConnected(previous_node, next_node) and previous_node is not None:
        messagebox.showwarning("Warning!", "Can't travel to the selected room!")
    else:
        self._select_artist(artist)
        self.visited.append(next_node)
        if len(self.visited) == self.no_vertices and not self.isSolved():
            messagebox.showwarning("Warning!", "You didn't pass through the marked edge! Resetting...")
            self.deselect_all_artists()

```

Slika 33. Klik na novi vrh grafa

self._selected_artists predstavlja listu svih označenih vrhova, a *artist* predstavlja vrh koji se trenutno provjerava. Najprije se provjerava je li duljina liste označenih vrhova veća od 0. Ako je, tada tražimo koji je vrh bio prethodni, odnosno koji je zadnji koji je bio posjećen prije novog kliknutog, te mu mijenjamo boju u crnu. Funkciji *_get_node_by_artist* predajemo objekt iz *matplotlib.patches* modula, stvarni nacrtani krug. Funkcija vraća indeks vrha koji je zapravo ključ za zadani krug. Slijedi provjera jesu li prethodno kliknuti

i najnovije kliknuti vrhovi povezani. Ako nisu, prikazat će se upozorenje s porukom da je nemoguće doći u željenu sobu (vrh). Inače se poziva funkcija za označavanje vrha (mijenja boju vrha u zelenu te ga dodaje u *self._selected_artists*), indeks tog vrha se dodaje u listu posjećenih, te se radi zadnja provjera. Ukoliko je duljina liste posjećenih jednaka ukupnom broju vrhova, ali graf još nije riješen, prikazuje se upozorenje s porukom da igrač nije prošao označenim bridom, nakon čega se svi vrhovi odznače tako da se svima promijeni boja u bijelu, a liste posjećenih i označenih vrhova se isprazne.

Algoritam za pronalazak Hamiltonovih putova

Svaki graf odmah prilikom inicijalizacije pronaći će sve moguće Hamiltonove putove koje prolaze označenim bridom (slika 32 ne pokazuje kôd nakon inicijalizacije članske varijable *hamiltonian_paths*). Algoritam koji se koristi je zapravo iscrpna pretraga uz tzv.

backtracking. Za svaki brid iz neposjećenog vrha provjeravamo vodi li rješenju ili ne. Slika 34 prikazuje kôd koji pronalazi sve Hamiltonove putove koji počinju iz jednog vrha.

```

2314 def find_hamiltonian_paths(self, start, visited, path):
2315     if len(path) == self.no_vertices and tuple_in_list(self.marked_edge, path):
2316         self.hamiltonian_paths.append(path[:])
2317         return
2318
2319     for i in self.adjacency_matrix[start]:
2320         if not visited[i]:
2321             visited[i] = True
2322             path.append(i)
2323
2324             self._find_hamiltonian_paths(i, visited, path)
2325
2326             visited[i] = False
2327             del path[len(path) - 1]

```

Slika 34. Backtracking algoritam za Hamiltonove putove

start označava indeks vrha iz kojeg počinjemo. *visited* je lista trenutno posjećenih vrhova, a *path* je lista trenutne rute. S obzirom da je rješenje rekurzivno, odmah na početku se provjerava je li duljina trenutne rute jednaka ukupnom broju vrhova grafa i postoji li u trenutnoj ruti označeni brid. Ako su oba uvjeta zadovoljena, trenutna ruta se dodaje u člansku varijablu *hamiltonian_paths*. U suprotnom, za svaki brid koji počinje od vrha *start* provjeravamo postoji li rješenje. Procesiramo samo neposjećene vrhove (jer trebamo proći svakim vrhom točno jednom). U liniji 2324 rekurzivno provjeravamo donosi li dodavanje vrha *i* u trenutnu rutu (*path*) rješenje. Naposljetku se događa vraćanje, tj. *backtracking*.

Primjećujemo da ovaj algoritam pronalazi sve Hamiltonove putove samo iz jednog vrha. Slika 35 prikazuje dio kôda u inicijalizaciji grafa gdje se provodi funkcija sa slike 34 za svaki vrh.

```
2213     for i in range(self.no_vertices):
2214         visited = [True if j == i else False for j in range(self.no_vertices)]
2215         path = [i]
2216         self._find_hamiltonian_paths(i, visited, path)
```

Slika 35. Traženje Hamiltonovih putova za svaki vrh

Naposljetku, funkcija za provjeru je li graf riješen vrlo je jednostavna. Prikazana je na slici 36.

```
2337     def isSolved(self):
2338         return self.visited in self.hamiltonian_paths
```

Slika 36. Funkcija za provjeru rješenja

Funkcija samo ispituje postoji li takav put kojim se igrač kretao (lista posjećenih vrhova) u listi svih Hamiltonovih putova toga grafa.

4.5 Povezanost *PlayableGraph* objekta sa sučeljem i *MainApp*-om

Prilikom svakog klika mišem na podlogu, osim uvjeta koji su se provjeravali prikazanih na slici 33, provjeravat će se je li graf riješen (slika 36). Slika 37 prikazuje kôd koji se izvodi u slučaju da je graf riješen.

```
307     def _on_press(self, event):
308         if self.plot_instance.isSolved():
309             self.controller.points += self.plot_instance.no_vertices
310             self.points_label.config(text="Current points: {}".format(self.controller.points))
311             if self.difficulty == "medium":
312                 self.time += 3
313             elif self.difficulty == "hard":
314                 self.time += 5
315             self.nextGraph()
```

Slika 37. Pritisak mišem kad je graf riješen

plot_instance predstavlja instancu trenutnog *PlayableGraph* objekta. Ako je riješen, bodovima se dodaje iznos ukupnog broja vrhova grafa te se osvježava labela koja prikazuje trenutni ukupan broj bodova. Zatim se dodaje vrijeme ovisno o razini igre. Naposljetku se poziva funkcija za sljedeći graf. Dio te funkcije prikazan je na slici 38.

```

286     def nextGraph(self):
287         self.plot_instance.deleteGraph()
288         if self.difficulty == "easy":
289             total_nodes = randint(4, 5)
290         elif self.difficulty == "medium":
291             total_nodes = randint(6, 7)
292         else:
293             total_nodes = randint(8, 9)
294         graph = create_graph(total_nodes, self.difficulty)
295         self.plot_instance = netgraph.PlayableGraph(graph)

```

Slika 38. Funkcija za sljedeći graf

Najprije se briše trenutni graf. Potom se, ovisno o razini igre, uzima broj vrhova sljedećeg grafa. Maksimalan broj vrhova grafa ne prelazi 9. Glavni razlog tomu je interaktivnost. Osim što bi grafovi bili puno kompliciraniji, bili bi i podložniji manje kvalitetnom iscrtavanju, te bi računalu mnogo dulje trebalo za pronaći sve Hamiltonove putove s obzirom da je složenost za iscrpnu pretragu faktoriijalna. Funkcijom *create_graph* stvara se matrica susjedstva koja služi kao argument za inicijalizaciju nove instance *PlayableGraph*-a.

Što se tiče gumba za nerješivost i naredbe koja se poziva pritiskom na njega, kôd je vrlo jednostavan. Prikazan je na slici 39.

```

316     def pressUnsolvable(self):
317         if not self.plot_instance.hamiltonian_paths:
318             self.controller.points += 2
319             self.points_label.config(text="Current points: {}".format(self.controller.points))
320             self.nextGraph()
321         else:
322             messagebox.showwarning("Fail!", "Given maze is solvable! Time penalty -2!")
323             self.time -= 2

```

Slika 39. Naredba za gumb za nerješivost grafa

Uvjet samo provjerava je li lista Hamiltonovih putova grafa prazna. Ako je, onda se dodaju bodovi i odmah se prelazi na sljedeći graf. U suprotnom, prikazuje se upozorenje s porukom da je graf rješiv, te se igraču oduzimaju dvije sekunde.

Na kraju, slika 40 prikazuje funkciju koja se izvodi prilikom pritiska na gumb za pomoć.

```

355     def click_help(self):
356         self.controller.points = 0
357         self.points_label.config(text="Current points: 0")
358         hamiltons = self.plot_instance.hamiltonian_paths
359         visited = self.plot_instance.visited
360         for path in hamiltons:
361             if visited == path[:len(visited)]:
362                 messagebox.showinfo("Information", "You're on the right track!")
363                 return
364         messagebox.showwarning("Warning", "Current path can't be right! Try undoing/resetting.")

```

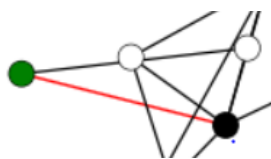
Slika 40. Naredba za gumb za pomoć

Najprije se bodovi ažuriraju na nulu. Zatim se za svaki Hamiltonov put grafa provjerava sadrži li trenutnu posjećenu rutu. Brzo i jednostavno *slice*-anje listi u Pythonu ovdje nam prilično olakšava posao (*path[:len(visited)]*). Ukoliko trenutna posjećena ruta postoji u nekom od rješenja, otvorit će se prozor s informacijom da je igrač na pravom putu. Inače će se otvoriti prozor s upozorenjem da trenutna ruta ne može dovesti do rješenja, te da igrač mora pokušati resetirati ili poništiti zadnji potez.

5. Zaključak

Ono što je najbitnije za računalnu igru je da nije dosadna, i da je igrač želi igrati više puta (*replayability*). *Lewis in a Maze* to postiže kompetetivnošću, odnosno tablicom s najboljim rezultatima. Igra je prilično intuitivna i jednostavna za shvatiti, pa ipak neki grafovi (labirinti) mogu biti vrlo komplicirani. Odbrojavanje vremena uvelike pridonosi dinamičnosti igre, kao i činjenica da neki labirinti nemaju rješenja, zbog čega igrač treba biti osobito oprezan.

Postoje još neki elementi koji bi se mogli popraviti i nadodati. Ono što najviše smeta (a ujedno je i najkompliciranije za popraviti) je iscrtavanje grafova. Iako Fruchterman-Reingoldov algoritam radi prilično dobro, moguće su situacije u kojima su dva vrha povezana, a upravo između njih, baš na tom bridu nalazi se još jedan vrh. To može zbuniti igrača, jer može pomisliti da je taj vrh između dva spojena, povezan s jednim od njih, a možda i s oba. Slika 41 prikazuje jedan takav slučaj.



Slika 41. Slabo vidljiva nepovezanost vrhova

Ako se bolje pogleda, može se primijetiti da vrh u sredini nije povezan ni s desnim ni s lijevim (zelenim) vrhom. S obzirom da je igra vremenski ograničena, te igrač zapravo mora žuriti ako hoće postići dobar rezultat, ovakve situacije mogu biti frustrirajuće jer mogu dovesti do gubitka vremena zbog takvog zbunjujućeg crtanja grafa.

Još bi bilo zgodno implementirati mogućnost igraču da može nacrtati svoj graf, spremiti ga u bazu, te onda igrati s tako spremljenim grafovima iz baze. Time bi se dodatno iskoristile brojne mogućnosti *netgraph* biblioteke, a igrači bi mogli, osim brzine snalaženja i rješavanja problema, pokazati i određenu vrstu kreativnosti. Međutim, gore opisani problem s iscrtavanjem mogao bi ostati isti, posebno ako bi neki igrači namjerno pokušali eksploatirati taj problem, crtajući takve grafove u kojima izgleda kao da su neki vrhovi povezani dok zapravo nisu. Slika 42 pokazuje jedan takav primjer.



Slika 42. Primjer crtanja vlastitog grafa

Slika 42 prikazuje jedan interaktivni graf (nije objekt *PlayableGraph*). Nijedan od sivo označenih vrhova nije povezan ni sa čime, što bi moglo biti vrlo zbunjujuće za igrača.

Osim tog nedostatka, sve ostalo što se tiče igre funkcioniра upravo onako kako je i bilo zamišljeno.

6. Literatura

<https://www.fer.unizg.hr/>

<https://github.com/paulbrodersen/netgraph>

<https://web.math.pmf.unizg.hr/nastava/komb/predavanja/predavanja.pdf>

https://en.wikipedia.org/wiki/Hamiltonian_path

https://en.wikipedia.org/wiki/Hamiltonian_path_problem

<https://www.slideshare.net/KornepatiSeshagiriRao/np-complete-problems-in-graph-theory>

https://en.wikipedia.org/wiki/Graph_theory

<https://en.wikipedia.org/wiki/NP-completeness>

<https://docs.python.org/3.4/library/sqlite3.html>

<https://matplotlib.org/3.1.0/index.html>

<https://gordonlesti.com/use-tkinter-without-mainloop/>

<https://en.wikipedia.org/wiki/Tkinter>

https://en.wikipedia.org/wiki/Force-directed_graph_drawing

<https://www.r-graph-gallery.com/247-network-chart-layouts/>

Računalna igra zasnovana na potrazi za hamiltonovskim ciklusom u neusmjerenom grafu

Sažetak

Lewis in a Maze jednostavna je računalna logička igra napisana u programskom jeziku Python. Cilj igre je u zadanom vremenu riješiti što više slučajno nacrtanih grafova, na način da se, za svaki od njih, pronađe jedan Hamiltonov put. Ako takav put ne postoji u grafu, potrebno je pritisnuti gumb za nerješivost grafa. Postoje tri različite razine koje igrač može odabrati. O tim razinama ovisi broj vrhova grafova koje treba riješiti, kao i koeficijent kojim se na kraju igre množi ukupan broj bodova. Rezultati se mogu spremati u tablicu s rezultatima (bazu podataka, *highscores*).

Grafičko sučelje igre realizirano je Pythonovom standardnom bibliotekom *tkinter*, a tablica s rezultatima standardnom bibliotekom *sqlite3*. Što se tiče crtanja grafova, iskorištena je i modificirana vanjska biblioteka *netgraph*: *open-source* projekt s *github* repozitorija za upravljanje, stvaranje i prikazivanje grafova.

Glavni algoritam koji se koristi za pronalazak svih Hamiltonovih putova u grafu je iscrpna pretraga s *backtrackingom*.

Ključne riječi: Računalna igra, logička igra, teorija grafova, neusmjereni grafovi, bestežinski grafovi, hamiltonovost, Hamiltonov ciklus, Hamiltonov put, iscrpna pretraga, backtracking, Python, tkinter, matplotlib, numpy, sqlite3

Computer game based on Hamiltonian cycle search in an undirected graph

Abstract

Lewis in a Maze is a simple computer puzzle game written in a programming language called Python. Goal of the game is to solve as many graphs as possible, in the given time. To solve a graph means to find a Hamiltonian path in it. If such a path doesn't exist, it is possible to state so by pressing the „Unsolvable“ button. There are three difficulties to choose. Depending on those difficulties, the number of nodes in a graph varies, as well as the factor which multiplies total collected points at the end of the game. Results can be saved to table with the highscores (database file).

Graphic user interface (GUI) is implemented with the Python's standard module, *tkinter*, while the highscores are implemented with *sqlite3*. As for graph plotting, an external library called *netgraph* is used and modified. It is an open-source project from the *github* repository for managing, creating and plotting (displaying) graphs.

The main algorithm used for finding all the Hamiltonian paths in the graph is the *brute-force* algorithm with backtracking.

Keywords: Computer game, puzzle game, graph theory, undirected graphs, weightless graphs, Hamiltonian cycle, Hamiltonian path, brute-force search algorithm, backtracking, Python, tkinter, matplotlib, numpy, sqlite3