SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 6476

PRIMJENA ALGORITAMA TRAŽENJA PUTA U GRAFU PRI RAZVOJU JEDNOSTAVNE 3D VIDEOIGRE

Marko Grozaj

Zagreb, veljača 2020.

Sadržaj

1 Uvod 4

2 Povijest videoigara i osnove dizajna videoigara 5

2.2 Povijest videoigara 5

2.3 Razvoj videoigara 7

3 Algoritmi traženja puta u grafu 10

3.1 Uvod u algoritme traženja puta u grafu 10

3.1.1 Breadth First Search 10

3.1.2 Depth First Search 11

3.1.3 Heuristika 11

3.1.4 Najbolji prvi 12

3.1.5 Algoritam A* (engl.A-star) 14

3.2 Usporedba algoritama za traženje puta u grafu 18

4.Dizajn videoigre 22

4.1 Cilj 22

4.2 Akcije 22

4.3 Izazovi 22

4.4 Interakcije 23

4.5 Modeli 23

4.6 Tijek igre 23

5 Razvoj videoigre 25

5.1 Razvojna okolina 25

5.2 Alati 25

5.2.1 Unity 25

5.2.3 Blender 27

5.3 Izvedba modela 28

5.4Izvedba interakcija 30

5.5 Primjena algoritma za traženje puta 34

6 Rezultati.....37

7 Zaključak.....40

8 Literatura.....41

1 Uvod

Avanturističke igre temelje se na interaktivnoj priči u kojoj igrač kao glavni lik istražuje virtualni svijet, nadilazeći pritom razne prepreke i rješavajući zadaće računalno-generiranih likova. Oblikovanje takve igre predstavlja brojne izazove koji se rješavaju primjenom znanja iz područja virtualnih okruženja, interaktivne računalne grafike, algoritama i struktura podataka te razvoja programske podrške. Krajnji cilj je imati osmišljenu i izvedenu računalnu igru u programskom okviru Unity te primijeniti algoritam za traženje puta u grafu na računalno generiranim likovima. Igra treba uključivati kretanje igrača kroz fizikalno zasnovani virtualni svijet, interakciju s istim te interakciju s računalno generiranim likovima koji slijede i napadaju igrača kroz zamišljeni virtualni svijet. Rad je organiziran u 6 poglavlja gdje se opisuje postupak izrade videoigre od istraživačkog djela do znanja primijenjenog pri samoj izradi videoigre.

2 Povijest videoigara i osnove dizajna videoigara

Igra je aktivnost igranja, zabave, složena u kontekstu izmišljene stvarnosti gdje igrač mora postići neki cilj. Videoigra je oblik igre gdje se aktivnost odvija na računalu ili igraćoj konzoli priključenoj na TV.

2.2 Povijest videoigara

Prosječni korisnik, u nastavku igrač, od moderne videoigre ima visoka očekivanja: od toga da igra mora biti vizualno lijepa, imati dobru priču, uroniti igrača u neki drugi svijet, pružiti mu sate zabave. Zato je razvoj igre vrlo zahtjevan. Dizajn videoigre obuhvaća razne vještine poput modeliranja, animacija, pisanja scenarija, pisanja koda. U početku razvoja videoigara sve je to bilo drugačije. Sredinom dvadesetog stoljeća nekoliko kompanija uvidjelo je da će arkadne igre promijeniti zabavu kakva je do tad bila poznata. Među njima, vjerojatno najinteresantnije, jer su zaista utjecale na razvoj videoigara ,su „Nintendo“ , „Sega“ te „Atari“. SEGA je 1966. izdala arkadnu igru „Periscope“.

Slika 1 Arkadna igra Periscope

Cilj te igre bio je jednostavno oboriti što više brodova. Igra je ubrzo postala vrlo popularna u Japanu pa su je ubrzo počeli izvoziti i u zapadne zemlje.
Jednostavnost je u ranim igrama bila ključ. Razlog zašto je jednostavnost bila ključ uspjeha leži u pristupačnosti svakoj osobi odnosno igra je intuitivna i svaka osoba, bez obzira na dob, može njoj pristupiti i znati što treba raditi kako bi se zabavila. U razdoblju kasnih sedamdesetih i ranih osamdesetih, proizvođači su uvidjeli kako se u videoigrama krije velika zarada i počeli su smišljati nove načine kako privući kupce da počnu što više igrati igre. Komercijalno veliki proizvođači su igre uglavnom plasirali na arkadne mašine koje su bile relativno skupe i nepristupačne

velikom broju korisnika koji bi ih htjeli imati kod kuće. Iz tog razloga počeli su igre prebacivati na takozvane kućne konzole. Kućne video konzole bile su praktične i lagane u usporedbi s arkadnim mašinama. Glavna karakteristika kućnih konzola jest interaktivni ekran. Tu ideju interaktivnog ekrana ili interaktivne televizije iznio je 1951. Ralph Baer no nema zapisa da je tu ideju uspio progurati tako da se kućne konzole nisu pojavile na tržištu još 20 godina. Prva kućna konzola zvala se Magnavox Odyssey koju je napravio Nintendo. Nije doživjela veliki uspjeh po pitanju prodaje no po pitanju patenta bila je izuzetno uspješna. Samo tri godine nakon toga pojavljuju se već dvije nove kućne konzole na tržištu: TV Tennis Electrotennis i poznatija Atarijeva konzola Atari Home Pong. Atarijeva konzola implementirala je njihovu najpoznatiju igru Pong koja se igra sa dvije palice za upravljanje i jednom lopticom. Cilj igre je spriječiti lopticu palicama da ode izvan granica jer u protivnom protivnik osvaja bod. Atari je izdavanjem te konzole postao jedna od vodećih tvrtki u razvoju videoigara. Paralelno s ovim pojavljivanjem kućnih konzola događaju se i promjene na polju kompleksnosti videoigara. Igre koje se razvijaju dobivaju veću dimenziju od same zabave. Atari 1980. izdaje „Space Invaders“ igru koju mnogi i dan danas pamte. Nakon toga Nintendo izlazi sa svojom „Donkey Kong“ igrom koja je zapravo preteča njihovog super hita „Super Mario“. Te 1981. kada je nastao Donkey Kong, nastala je i jedna od najpopularnijih retro igara, a to je „Pacman“.Ta godina mogla bi se označiti kao prijelazna godina u razvoju videoigara. Igre počinju izgledati privlačno, počinje se razmišljati o pričama koje su u pozadini same igre, postoji sustav bodovanja koji potiče natjecanje među igračima. I upravo je to natjecanje među igračima ta nova dimenzija igara. Jednostavno, svatko želi biti bolji od drugih i time se stvara užitak u igranju igara. Vrlo je važno spomenuti 1983. godinu jer je ona također svojevrsno obilježila povijest videoigara zato što je sa prodajom krenuo „NES“ ili „Nintendo Entertainment System“. Kućne konzole su time postale pristupačne i svi su ih htjeli imati pogotovo zato što se na njima mogla igrati igra „Super Mario Bros“ koja se i danas smatra vrhunski odrađenom igrom što se tiče dizajna razina. Nakon uspjeha „NES“-a na tržištu se devedesetih godina pojavila Sonyjeva konzola „Playstation“. Pojavom Playstationa obilježeno je novo razdoblje za igre koje su bile sve naprednije i naprednije. Kako je tehnologija brzo napredovala tako je i sklopovlje za igranje igara postojalo sve sposobnije. U devedesetima bilo je teško predvidivo da ćemo danas imati sklopovlje koje će omogućiti izradu

videoigara sa takvim detaljima da se igraču čini kako se okolina videoigre nalazi u stvarnom svijetu. I tom karakteristikom realizma danas igre zauzimaju mjesto u oblikovanju nekih spoznaja igrača gdje je prije tome služio film. Ne samo da je napredovala tehnologija već su se i promijenili stavovi oko dizajna igre. U početku je cilj dizajnerima igara bio što više otežati igru s ciljem većeg profita. Igrač se danas smatra prijateljem kojeg treba zabaviti, a veliku ulogu u tome od početka ima sam dizajn.

2.3 Razvoj videoigara

Glavnina sadržaja u ovom poglavlju preuzeta je, odnosno parafrazirana, iz knjige Fundamentals of Game Design [2].

Razvoj videoigara uključuje dizajn istih te implementaciju. Dizajn videoigara zahtjeva širok spektar znanja. Obuhvaća mnoga polja znanosti, ali i umjetnosti i to spaja u jedinstvenu cjelinu koja zabavlja igrače. Dizajner videoigre odgovoran je za osmišljavanje igre, njezinih mehanika te za opisivanje svakog pojedinog elementa igre, od funkcionalnih do vizualnih. Videoigre spajaju umjetnost i inženjerstvo jer svaka igra mora imati određeni vizualni stil, ali također biti funkcionalna u smislu mehanika koje uključuju kretanje igrača, računalno generirane protivnike, sustav bodovanja i slično. Na samom početku proizvođač mora odrediti svrhu njegove igre. Mora odlučiti želi li da se igra proda u puno primjeraka ili želi pokazati sve sposobnosti današnje tehnologije ili ima neku drugu ideju. Prema tome igre možemo podijeliti na:

- Igre s ciljem velike količine prodanih primjeraka
- Igre s naglaskom na dizajn
- Igre s naglaskom na tehnologiju
- Igre s naglaskom na umjetnost

U svim navedenim tipovima igara, osim u igri orijentiranoj na umjetnost, dizajn orijentiran na igrača je jedna važnija filozofija prilikom dizajniranja videoigre. Filozofija usmjerena na igrača ima dvije bitne stavke kojih se dizajner videoigara treba pridržavati :

- Dužnost da zabavi
- Dužnost da razumije igrača

Dužnost da zabavi govori sama za sebe. Igrači kupuju videoigru s očekivanjem da će biti zabavna. Svaki aspekt videoigre za koji se smatra da igraču neće biti zanimljiv treba ili zanemariti ili ga dodatno urediti i prilagoditi. Dužnost da razumije igrača podrazumijeva „stavljanje dizajnera u cipele reprezentativnog igrača“. Reprezentativni igrač predstavlja skupinu ljudi za koju je ta igra namijenjena. Teško je zadovoljiti svakog igrača i nerealan je scenarij kako će svi biti oduševljeni svakim od elemenata igre, upravo se zato uzima i reprezentativni igrač. Prilikom određivanja reprezentativnog igrača mora se uzeti u obzir objektivnost. Ono što se sviđa samom dizajneru ne znači da će se svidjeti i igraču. Svaka vizualna i funkcionalna komponenta mora biti pomno razmatrana sa stajališta reprezentativnog igrača. Primjerice, tvrtka je dobila zadatak izraditi igru za djecu do 6 godina koja će biti jednako zabavna kao i edukativna. Dizajner se, dakle, „postavlja u cipele djeteta“, sigurno će odbiti elemente koji previše opterećuju korisničko sučelje, funkcionalnosti koje nisu intuitivne ili su previše komplicirane, a tema takve igre će uključivati neke dječje likove ili vrlo privlačan vizualni stil pun boja. Razumijevanje igrača bitan je dio razvoja videoigara jer utječe na kasniju prodaju i korištenje igre. Dizajn igre može se opisati kroz tri faze: faza ideje, faza obrade, faza modifikacije. Faza ideje uključuje samu ideju igre. Na koji će način igra zabaviti igrača, koja je svrha igre, definiranje kome je igra namijenjena te određivanje uloge igrača. Ideja koja je bila na početku se tijekom cijelog projekta ne bi trebala mijenjati kako bismo bili sigurni da ćemo dobiti dobru videoigru. Zatim slijedi faza obrade koja uključuje detalje same igre od modeliranja svijeta, modeliranja likova, izrade pojedinih razina, pisanje priče te izgradnja prototipa, testiranje. Faza modifikacije je posljednja faza u dizajnu jer više nema mogućnosti za dodavanjem novih funkcionalnosti, novih mehanika, novih komponenti već se sve postojeće funkcionalnosti igre testiraju i pokušava se naći i najmanja mana u dizajnu razine, dizajnu glavnog lika, dizajnu mehanika, kako bi igra bila gotovo savršena i funkcionirala onako kako je to bilo zamišljeno u fazi ideje. Dizajn igre najbolje je podijeliti na više manjih problema. Kvalitetna igra zahtjeva velik broj ljudi, koji će međusobno surađivati i otkrivati greške, nedostatke koje jedan čovjek sam ne bi uočio. Danas su igre vrlo napredne, mogu se mjeriti sa filmovima sa strane priče i glavnih likova, tako će suradnja u timu stvoriti kvalitetnu igru koja će pokriti sve zahtjeve publike za koju je namijenjena. Kao što je već navedeno, igra uključuje dizajn i razvoj, ali i testiranje. A tako možemo i definirati dvije grupe ljudi

ovisno o poslu kojim se bave na samoj igri, dizajnerski tim i razvojni tim. Dizajnerski tim već ima definirane svoje zadatke no i u samom timu postoje uloge koje omogućavaju raznovrsnost i kontrolu dizajna.

- „*Lead Designer*“ – vođa dizajn tima, nadgleda sve ostale pozicije u timu i brine se o konceptualnoj ideji igre.
- „*Game Designer*“ – definira glavne mehanike i funkcionalnosti igre
- „*Level Designer*“ – osoba zadužena za modeliranje razina i računalnog svijeta
- „*User Interface Designer*“ – osoba zadužena za izgled i poziciju korisničkog sučelja na ekranu
- *Pisac* – osoba zadužena za priču igre, opise glavnih i sporednih likova
- „*Art Director*“ – osoba zadužena za vizualni koncept igre, modele likova, okruženja
- „*Audio Director*“ – osoba zadužena za slušni dio igre, odnosno dijaloge, glazbu, ambijentalne zvukove i slično

Svaka od ovih uloga može se dodijeliti jednoj ili više osoba, opet ovisno o njhovim zanimanjima, vještinama i sposobnostima. Razvojni tim, s druge strane čine programer koji svaku od ideja ljudi zaduženih za dizajn implementiraju. Tim dizajnera i razvojni tim usko su povezani i konstantno rade s timom testera kako bi uočili pogreške u prototipu, a kasnije i u gotovoj igri.

3 Algoritmi traženja puta u grafu

3.1 Uvod u algoritme traženja puta u grafu

Mnogo analitičkih problema zahtjeva sustavno pretraživanje prostora stanja. U rješavanju problema postoji početno stanje, a algoritam treba pronaći put do cilja. Početak svakog rješavanja problema jest definicija problema. Definiramo skup u kojem se nalazi svako moguće stanje u kojem se problem može naći. Također na početku algoritma moramo definirati početno stanje, iz kojeg ćemo krenuti sa izvedbom algoritma i pretraživanjem, te ciljno stanje koje nam označava kraj algoritma. Osnovna ideja pretraživanja je sljedeća:

- Ako se ne nalaziš u ciljnom stanju, nađi susjede
- Spremi u listu posjećenih, odaberi jednog susjeda i napravi tranziciju
- Ponavljaj dok ne dođeš na cilj

Prilikom korištenja algoritama traženja puta u grafu idealno bi bilo kada bi koristili algoritam koji je optimalan i potpun. Optimalan algoritam pronalazi optimalan put do cilja, onaj put koji s najmanjom cijenom puta ili ne zapinje u nekim petljama. Potpun algoritam prilikom svakog izvođenja uvijek pronalazi rješenje ako ono postoji. Kako bi primijenili neki od algoritama trebamo imati definirano stablo pretraživanja. Stablo pretraživanje kao korijen ima početno stanje, a čvorovi su njegovi susjedi. Obzirom na pretraživanje stabla imamo dvije strategije pretraživanja koje imaju nekoliko tehnika pretraživanja.

- Slijepo pretraživanje – slijepo ili neinformirano pretraživanje ne koristi nikakve dodatne informacije o čvorovima osim onih koje su definirane na početku problema(npr. *BFS*,*DFS*,*IDS*)
- Heurističko pretraživanje – heurističko ili informirano pretraživanje koristi dodatne informacije o čvorovima, konkretno koristi iskustvenu (heurističku) funkciju koja procjenjuje put od sljedećih stanja do cilja.(npr. *A**,*najbolji prvi*)

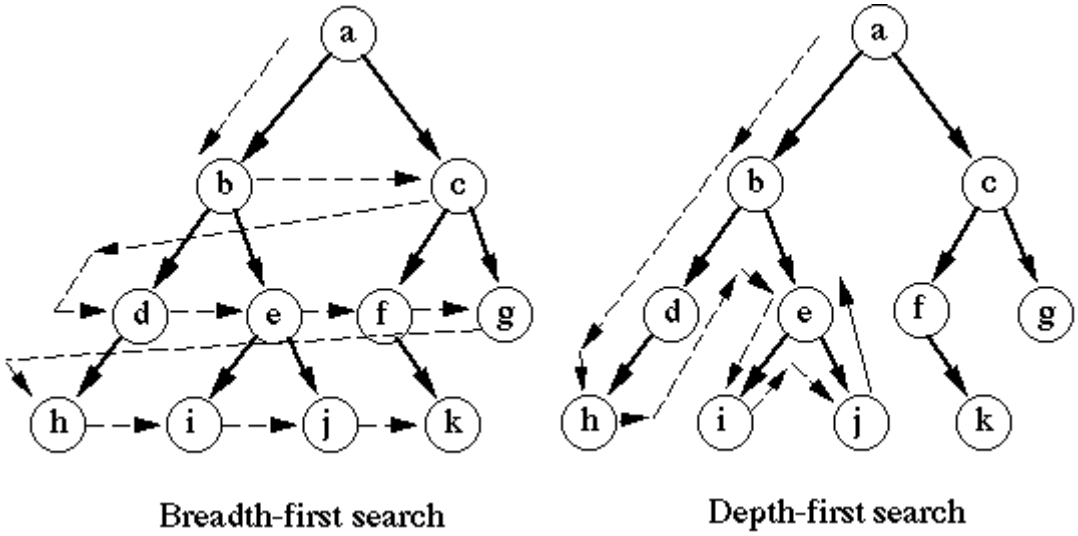
3.1.1 Breadth First Search

Pretraživanje u širinu (engl**. Breadth first search**, skr**.BFS**) nakon korijenskog čvora proširuje svu njegovu djecu. BFS se koristi kod manje složenih problema jer iziskuje veliku količinu memorije za svoje izvođenje. Malom modifikacijom BFS-a

dobivamo pretraživanje s jednolikom cijenom koje uzima u obzir cijene prijelaza. Oba ova algoritma su i optimalna i potpuna.

3.1.2 Depth First Search

Pretraživanje u dubinu (engl. **Depth first search,** skr. **DFS**) prvo proširuje najdublji čvor i tako sve dok ne dođe do listova stabla, čvorova koji nemaju djece. Praktična izvedba DFS-a je stog jer generirane čvorove dodajemo na početak liste neposjećenih. Ovaj algoritam nije ni potpun, a ni optimalan. Razlog što nije potpun je da može zapeti u petlji i nikada ne doći do cilja, a razlog zašto nije optimalan jest taj što ne pretražuje razinu po razinu. DFS bi trebalo izbjegavati kod problema gdje stablo pretraživanja ima veliku ili beskonačnu maksimalnu dubinu. Modifikacijom DFS-a nastaje ograničeno pretraživanje u dubinu gdje idemo do određene dubine i zatim se vraćamo, ne idemo skroz do listova. Taj algoritam je potpun, ali ne i optimalan.



Slike 2 Usporedba algoritma BFS i DFS

Kombinacijom DFS-a i BFS-a dobivamo iterativno pretraživanje u dubinu (engl. **Iterative depth search**, skr. **IDS**) gdje proširujemo djecu trenutnog čvora, ali samo do određene dubine koju iterativno povećavamo. Taj je algoritam i optimalan i potpun.

3.1.3 Heuristika

Za shvaćanje principa algoritama koji koriste strategiju heurističkog pretraživanja treba znati pojam heurističke funkcije ili iskustvene funkcije. Heuristiku definiramo kao iskustvena pravila o prirodi problema i osobinama cilja čija je svrha

pretraživanje brže usmjeriti prema cilju. Heuristička funkcija pridjeljuje svakom stanju procjenu udaljenosti od tog stanja do ciljnog stanja. Što je ona manja to je čvor bliže kraju, a ako je je njen iznos nula onda smo došli u ciljno stanje. U definiciji problema heuristička se funkcija označava s ***h(s)*** gdje **s** predstavlja stanje. Algoritme koji koriste heurističke funkcije nazivamo još i usmjerenim algoritmima. Svaka heuristiku možemo ocijeniti pomoću tri karakteristike, svojstvo *konzistentnosti*, svojstvo *dominacije* i *optimističnost*. Ocjenjivanjem heuristike možemo odlučiti hoće li ona odgovarati za rješavanje problema ili ne. Svaka heuristika koja ne precjenjuje odnosno nikad procjena nije veća od stvarne cijene puta karakterizira se kao optimistična heuristika. Svojstvo dominacije govori koliko je neki algoritam obavještеniji od drugog. Obavještеniji algoritam pretražuje manji broj stanja od manje obaviještenog algoritma.

SVOJSTVO DOMINACIJE

Neka su A i B dva optimalna algoritma s optimističnim heurističkim funkcijama h1 i h2. Algoritam A dominira nad algoritmom B ako:

$$\forall s \in S,\ h1(s) \geq h2(s)$$

Ako je heuristika konzistentna, znači da je nužno optimistična, a to znači da prvi generirani čvor je i čvor s najmanjom cijenom pa ne moramo provjeravati cijenu već zatvorenih čvorova.

SVOJSTVO KONZISTENTNOSTI

$$\boldsymbol{f(n1) \leq f(n2)}$$

$$\boldsymbol{g(n1) + h\big(state(n1)\big) \leq g(n2) + h(state(n2))}$$

$$\boldsymbol{g(n1) + h\big(state(n1)\big) \leq g(n1) + c + h(state(n2))}$$

$$\boldsymbol{h\big(state(n1)\big) \leq h\big(state(n2)\big) + c}$$

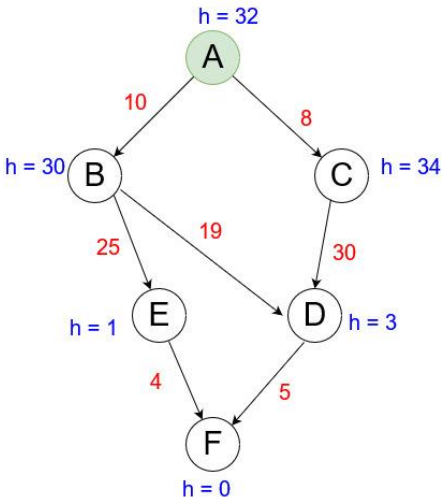
Heuristika h je konzistentna ili monotona ako:

$$\forall (s2,c) \in succ(s1),\quad h(s1) \leq h(s2) + c$$

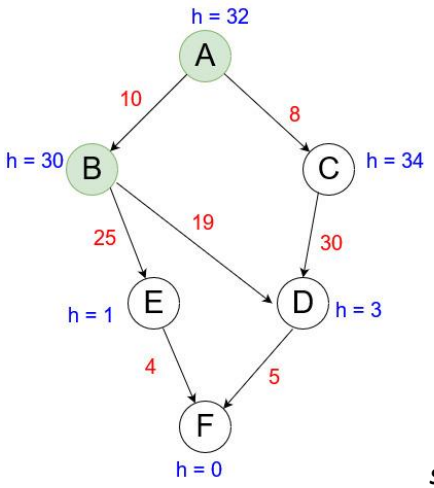
3.1.4 Najbolji prvi

Pohlepan algoritam ili **najbolji prvi** odabire onaj čvor koji se čini najbliži cilju. Koristi heuristiku kao izbor za sljedeći čvor, ukoliko ima dva ili više susjeda,

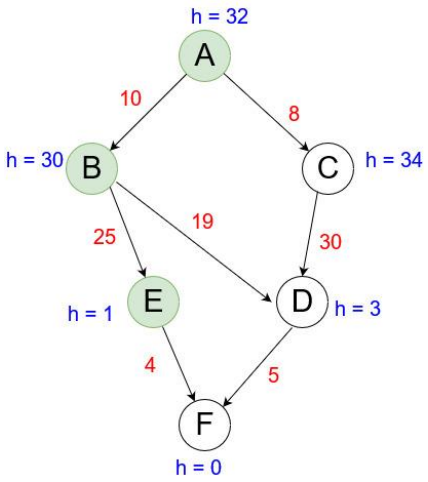
odabire onaj čija je heuristika najmanja smatrajući da je heuristika vrlo precizna i da mu daje točne podatke. No, heuristika nikada ne može biti jedini indikator cijene puta jer je to procjena udaljenosti između dva čvora. Pohlepan algoritam nije ni potpun ni optimalan. Na slikama je naveden i primjer zašto nam pohlepan algoritam nije povoljan. Početno stanje iz kojeg pokrećemo algoritam jest A, a ciljno stanje jest F.



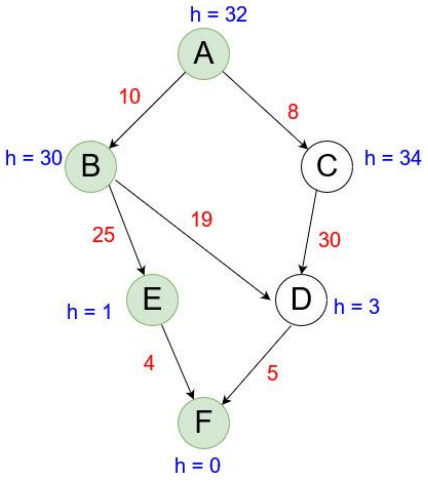
Slika 3 Početni čvor pohlepnog algoritma



Slika 4 Proširivanjem početnog čvora dobivamo dva djeteta od kojeg se odlučujemo za lijevo jer ima manju vrijednost heuristike



Slika 5 Odabiremo čvor koji ima manju heuristiku



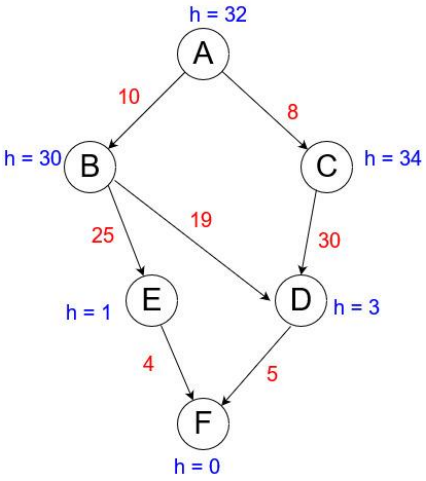
Slika 6 Dolazimo do cilja, ali ne optimalnim putem, umjesto da nas je put koštao 34, ovim putem nas je koštao 39

3.1.5 Algoritam A* (engl.A-star)

Za razliku od pohlepnog algoritma algoritam A*(engl. A-star, prevedeno A-zvjezdica algoritam) odabire sljedeći čvor na puno pametniji način. Pri odabiru sljedećeg čvora ne oslanja se samo na heuristiku koja procjenjuje udaljenost do cilja već i u obzir uzima cijenu puta. Dakle možemo zaključiti da je algoritam A* kombinacija algoritma traženja puta s jednolikom cijenom i najboljeg prvog. Funkciju kojom A* određuje koji će čvor uzeti sljedeći nazvat ćemo ukupna cijena i označit ćemo ju s *f(n)*.

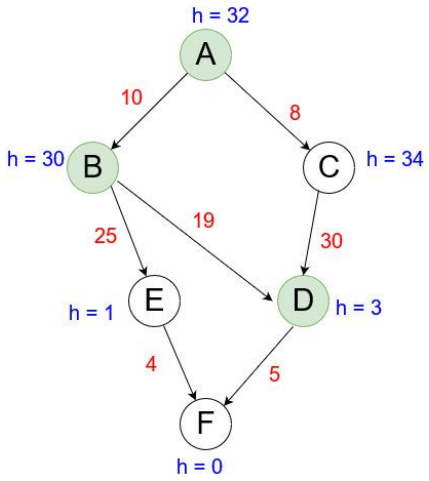
$$\boldsymbol{f(n) = g(n) + h(state(n))}$$

G(n) je stvarna cijena puta od početnog čvora do čvora n, a h(s) je procjena cijene udaljenosti od stanja s do cilja ili heuristička funkcija. Algoritam A zvjezdica uzima najbolje od oba svijeta (slijepog i heurističkog pretraživanja), optimalan je pod uvjetom da je heuristička funkcija optimistična te je potpun. Na primjeru grafa (*Slika 7*) provest ćemo algoritam A*. Izvorni kod algoritma dan je u nastavku, u osnovi u obzir uzima dosadašnju cijenu puta, cijenu prijelaza te heuristiku i odabire čvor s najmanjom vrijednošću tog zbroja.

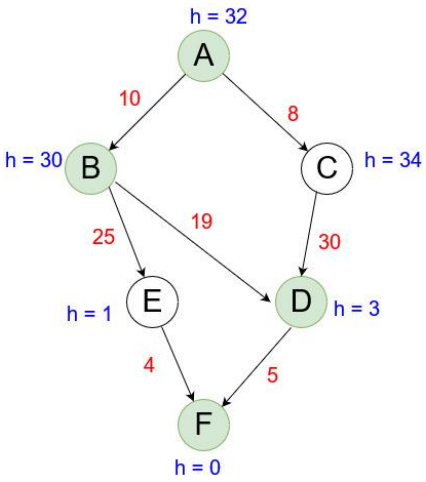


Slika 7 Fizička reprezentacija analiziranog grafa

Prva dva koraka su ista kao i kod algoritma najbolji prvi, dakle ulazimo u početni čvor A te odabiremo čvor B iz razloga jer zbroj cijene prijelaza i heuristike daje manju vrijednost od čvora C. No prilikom odabira sljedećeg stanja dolazi do promjene.



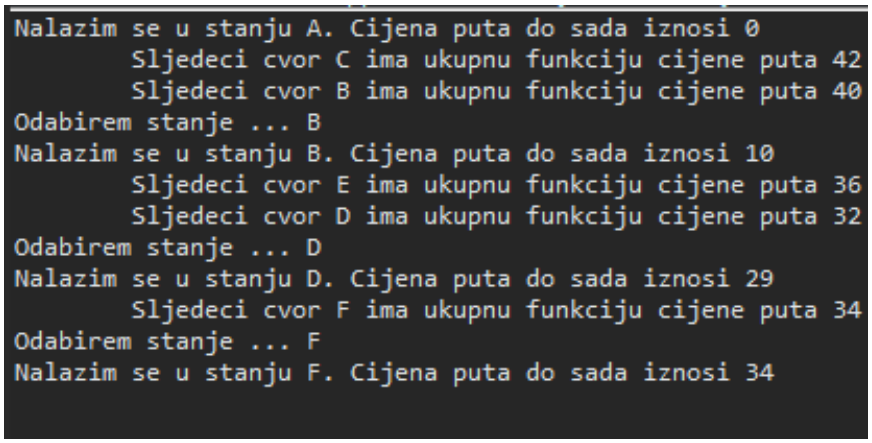
Slika 8 Odabiremo optimalniji put do cilja



Slika 9 Našli smo optimalan put do cilja s cijenom od 34

```
public class AStarAlgorithm {

	public void findPath(List<State> stanja) {
		State stanje = stanja.get(0);
		List<State> posjecenaStanja = new LinkedList<State>();
		int trenutnaCijena = 0;
		posjecenaStanja.add(stanje);
		ispis(stanje,trenutnaCijena);
		while(!stanje.getIme().equals("F")){
			HashMap<State, Integer> sljedećiČvorovi = stanje.getChildren();
			int najmanjiPut = Integer.MAX_VALUE;
			int udaljenost = 0;
			for(State state : sljedećiČvorovi.keySet()) {
				int ukupnaCijena = sljedećiČvorovi.get(state) + state.heuristic() +
trenutnaCijena;
				ispisDjece(state,ukupnaCijena);
				if(ukupnaCijena < najmanjiPut) {
					stanje = state;
					udaljenost = sljedećiČvorovi.get(state);
				}
			}
			System.out.println("Odabirem stanje ... " + stanje.getIme());
			trenutnaCijena += udaljenost;
			ispis(stanje, trenutnaCijena);
		}

	}

	public void ispis(State stanje, int trenutnaCijena) {
		System.out.println("Nalazim se u stanju " + stanje.getIme() + ". Cijena puta do sada
iznosi " + trenutnaCijena);
	}

	public void ispisDjece(State state, int ukupnaCijena) {
		System.out.println("	Sljedeci cvor " + state.getIme() +
					" ima ukupnu funkciju cijene puta " + ukupnaCijena);
	}

}
```

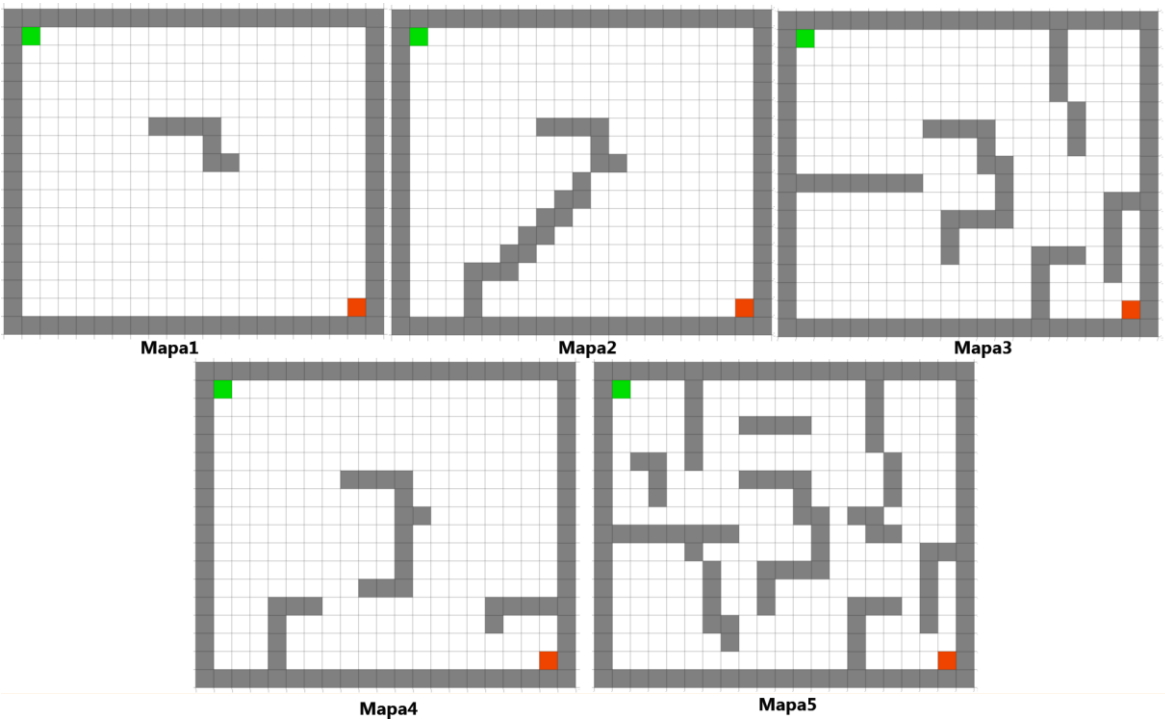
Slika 10 Algoritam A*



Slika 11 Rezultat izvođenja algoritma za graf sa slike 7

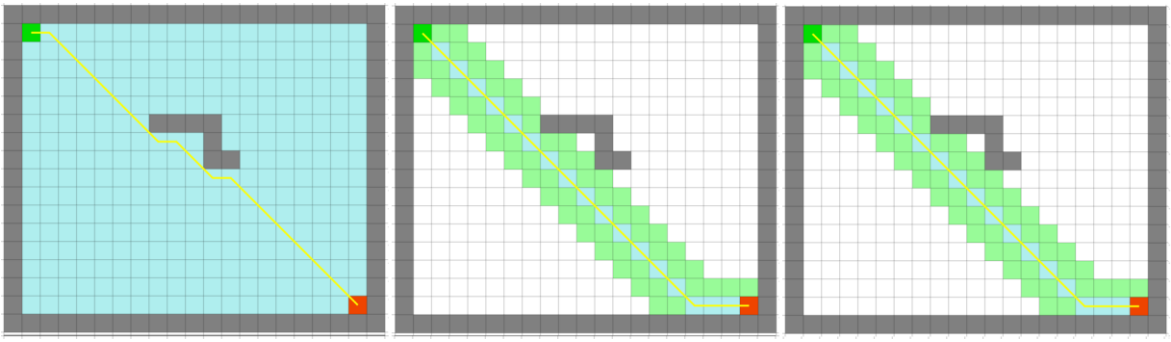
3.2 Usporedba algoritama za traženje puta u grafu

Algoritmi mogu biti povoljni i nepovoljni ovisno o situaciji u kojima ih koristimo. S obzirom na različita svojstva moramo analizirati svaki od algoritama kako bismo mogli odlučiti koji ćemo koristiti. Usporedbu možemo vršiti na temelju brzine izvršavanja, na temelju broja pretraženih stanja, na temelju brzine nalaženja puta, na temelju optimalnosti i slično. Kako bismo shvatili koje su prednosti algoritma A* napravit ćemo usporedbu sa BFS algoritmom te najboljim prvim. Kao test koristit ćemo https://qiao.github.io/PathFinding.js/visual/ gdje vrlo jednostavno možemo vizualno vidjeti na koji način funkcionira svaki od algoritama te možemo prilagođavati izgled mape. Promatrat ćemo trajanje izvršavanja algoritma te broj izvršenih operacija i duljinu pronađenog puta na 5 različitih mapa.



Slika 12 Mape za analizu algoritama

Na slici su prikazane mape nad kojima ćemo vršiti analizu algoritama. Mape su ručno modificirane kako bi se na svakoj sljedećoj povećala kompleksnog pronalaska puta i kako bi se prikazale prednosti svakog algoritma, ali ujedno i nedostatci. Dakle imamo tri algoritma koji se natječu u tri kategorije, svaka kategorija jednako je važna i prilikom ocjene algoritma svaka se kategorija uzima u obzir.

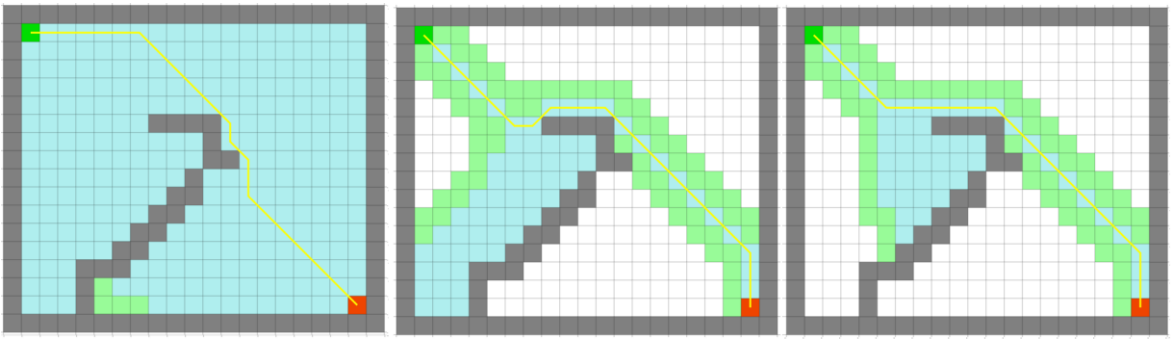


Slika 13 Rezultati izvršavanja BFS, najbolji prvi, A algoritma (Mapa 1)*

MAPA 1	BFS	NAJBOLJI PRVI	A*
Vrijeme izvršavanja [s]	8.32	1.91	1.31
Broj izvršenih operacija	594	98	98
Duljina pronađenog puta	24.21	24.21	24.21

Slika 14 Rezultati praćenih karakteristika (Mapa 1)

Valja primijetiti kako su sva tri algoritma našla optimalni put koji iznosi 24.21. BFS je poprilično spor u odnosu na druge algoritme i pretražio je sva moguća stanja te izvršio najveći broj operacija. Kada gledamo koji od algoritama, najbolji prvi i A*, je bolji vidimo da su oba izvršila jednak broja operacija i na slici vidimo da su pretražili isti broj stanja no A* je u tom izvršavanju ipak bio brži zbog heuristike gdje je odmah na početku mogao odbaciti neka stanja.



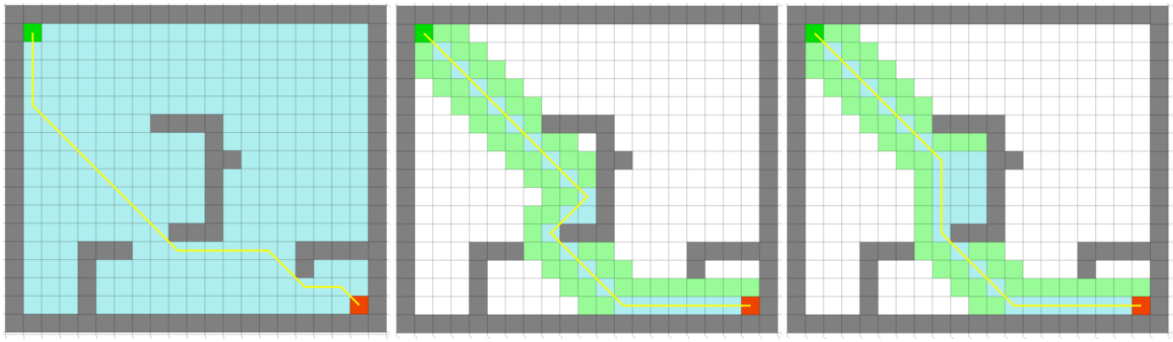
Slika 15 Rezultat izvršavanja BFS, najbolji prvi, A algoritma (Mapa 2)*

MAPA 2	BFS	NAJBOLJI PRVI	A*
Vrijeme izvršavanja [s]	7.98	2.94	2.89
Broj izvršenih operacija	562	199	158
Duljina pronađenog puta	25.97	26.8	25.97

Slika 16 Rezultati praćenih karakteristika (Mapa 2)

Na ovoj smo mapi napravili neke konkretnije prepreke što je pomoglo, uspoređujući s prvom mapom, algoritmu BFS tako što je trebao izvršiti manje operacija no i dalje je on uvjerljivo najgori. Ovoga puta algoritam najbolji prvi čak ni

nije pronašao optimalan put te je izvršio više operacija od algoritma A*. Također možemo primijetiti da se algoritam A* najbrže prilagodio kada je „shvatio“ da mu prepreka ne dozvoljava da prođe tim putem.

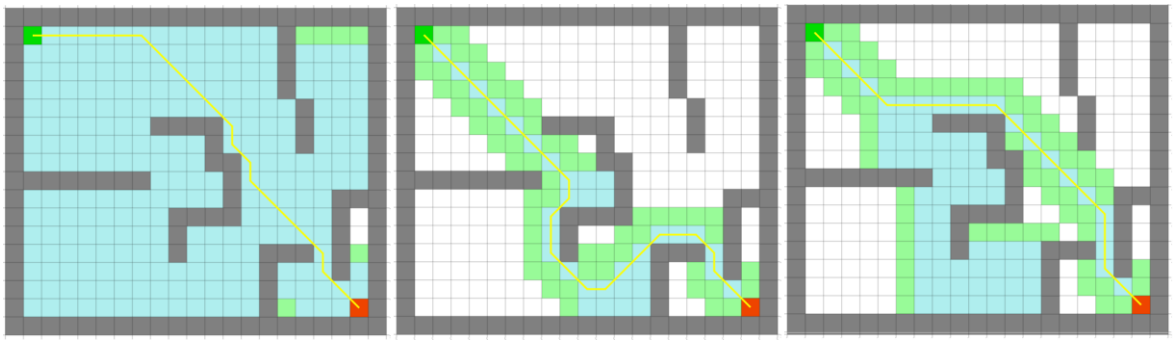


Slika 17 Rezultat izvršavanja BFS, najbolji prvi, A algoritma (Mapa 3)*

MAPA 3	BFS	NAJBOLJI PRVI	A*
Vrijeme izvršavanja [s]	8.36	1.70	1.51
Broj izvršenih operacija	560	104	113
Duljina pronađenog puta	26.56	28.21	26.56

Slika 18 Rezultati praćenih karakteristika (Mapa 3)

Ovom mapom pokazujemo da je bez obzira na sporost ponekad i bolje koristiti BFS obzirom da je on opet našao optimalan put za razliku od najboljeg prvog koji je uporno želio „prolaziti“ kroz zidove. I dalje je, bez sumnje, algoritam A* uvjerljivo najbolji, što se tiče i brzine i efikasnosti i optimalnosti.

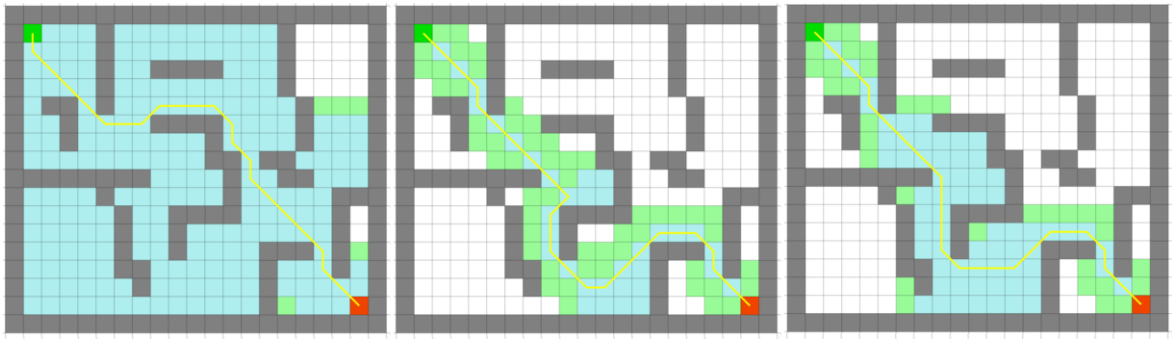


Slika 19 Rezultat izvršavanja BFS, najbolji prvi, A algoritma (Mapa 4)*

MAPA 4	BFS	NAJBOLJI PRVI	A*
Vrijeme izvršavanja [s]	8.19	2.42	3.54
Broj izvršenih operacija	516	131	217
Duljina pronađenog puta	25.97	31.04	25.97

Slika 20 Rezultati praćenih karakteristika (Mapa 4)

Ova mapa specifična je po tome što je uspjela usporiti algoritam A* po pitanju vremena izvršavanja, ali i po pitanju broja izvršenih operacija. No uzimajući optimalnost u obzir, vjerojatniji izbor bi opet bio algoritam A*.



Slika 21 Rezultat izvršavanja BFS, najbolji prvi, A* algoritma (Mapa 5)

MAPA 5	BFS	NAJBOLJI PRVI	A*
Vrijeme izvršavanja [s]	6.87	2.16	2.81
Broj izvršenih operacija	437	125	169
Duljina pronađenog puta	27.38	31.04	29.38

Slika 22 Rezultati praćenih karakteristika (Mapa 5)

Primjećujemo kako algoritam A* nije pronašao optimalan put za razliku od svih prijašnjih mapa, no to je normalno zbog prirode heuristike o kojoj ovisi koji će put algoritam izabrati.

Na koncu, algoritam A* od pet mapa na kojima je testiran optimalan put pronašao je četiri puta, dva puta izvršio najmanji broj operacija i tri puta je bio najbrži algoritam. Algoritam najbolji prvi od pet mapa na kojima je testiran optimalan put pronašao je samo jednom, tri puta izvršio najmanji broj operacija te dva puta bio najbrži u izvršavanju. Algoritam BFS od pet mapa na kojima je testiran optimalan put pronašao je svaki put, niti jednom nije izvršio najmanji broj operacija i nijednom nije bio najbrži. Zaključak koji možemo donijeti na temelju ovih testova jest da sve ovisi o potrebama. U konkretnom slučaju upotrebe algoritama za NPC (engl. *Non playable character*) želimo da gotovo uvijek nalazi optimalan put i da bude brz, a sve to najbolje zadovoljava algoritam A*.

4.Dizajn videoigre

Igra je, za potrebe, ovog rada nazvana „Mystery“. Glavni protagonist igre je robot futurističkog izgleda, koji se nalazi na njemu nepoznatom planetu gdje mu je cilj pronaći neobične pojave, stvari za određena područja. Nema nikakvih sjećanja o tome kako je došao ovdje i tko je on. Otkrivanjem specifičnih stvari on polako počinje otkrivati svoju priču i dobiva nagrade koje će mu pomoći da uspostavi kontakt sa svojom vrstom.

4.1 Cilj

Cilja igrača jest otkriti sve tajne odnosno nepoznate stvari kako bi robot shvatio tko je on i zašto se nalazi u ovom svijetu.

4.2 Akcije

Akcija	Opis	Važnost
Hodanje	Robot se pomiče u određenom smjeru brzinom hoda	Vrlo važno
Trčanje	Robot se pomiče u određenom smjeru brzinom trčanja	Važno
Napad	Napad na suprotnu stranu	Najvažnije

Hodanje je vrlo važna akcija u igri jer omogućuje kretanje igraču po svijetu pa tako i ostvarivanje krajnjeg cilja. Dvije su opcije kako bi igrač mogao kontrolirati hod robota u svijetu, jedna opcija su tipke W (kretanje naprijed), A (rotacija robota ulijevo, kretanje ulijevo), D (rotacija robota udesno, kretanje udesno) i odgovarajuća im druga opcija, strelica gore, lijevo i desno. Pritiskom tipke *Shift* robotu se mijenja brzina kretanja i prelazi u stanje trčanja. Akcija trčanja jest važna kako bi igrač mogao brže pobjeći čudovištu koje ga napada. Napad je akcija kojom se igrač brani od čudovišta i pokušava ga uništiti, pritiskom tipke *Space* aktivira se funkcija napada.

4.3 Izazovi

Postoje tri vrste izazova u igri. Svaki od ta tri izazova ima svoje značenje za igru. Izazov pronalaska tajni i rješavanja pitanja postojanja robota je najbitniji izazov jer tako igrač napreduje u igri. Prilikom istraživanja svijeta igrač će naići na ključeve,

što mu daje dodatan motiv za istraživanje svijeta kako bi pronašao sve ključeve. Također će u svijetu naići na čudovišta koja će ga napadati i pratiti sve dok ih ne uništi.

4.4 Interakcije

U svijetu se nalaze mjesta stvaranja čudovišta, igrač kada dolazi u dodir sa tim područjem, stvaraju se čudovišta na kojima je primijenjen algoritam za traženja puta te slijede igrača i dodirom sa njim uzimaju mu zdravlje. Istraživanjem svijeta igrač uvidi da postoje ključevi koje može pokupiti, svaki ključ na sebi ima okidač koji se aktivira ulaskom igrača u njegovo područje kolizije i aktivira uništenje i povećava broj sakupljenih ključeva.

4.5 Modeli

U svijetu se nalaze dvije vrste čudovišta. Plava čudovišta koja su statična i njihova jedina svrha jest pomoć protagonistu u trenutcima kada je to potrebno. Crvena čudovišta, s druge strane, imaju ulogu napadača igrača. Pokušavaju igraču otežati njegovu misiju. Svijet sačinjavaju 4 različita područja: šuma, područje prekriveno kamenjem, pustinja i park. Svako od tih područja sadrži specifične stvari, na primjer šuma sadrži puno grančica. Cilj je da igrač pronađe takve specifične stvari u drugim područjima, dakle grančice u pustinji ili kamenom području ili parku. Kako bi to napravio mora istražiti gotovo cijelo područje, pritom skupljajući ključiće.

4.6 Tijek igre

Na početku igre igrač se stvara na početnoj poziciji u parku. Na samom početku preko cijelog ekrana, s ciljem privlačenja pozornosti, pojavljuju se poruke koje igraču objašnjavaju koji je njegov cilj, kako se može kretati, koga se mora čuvati te mu daje početni „hint“ da igrač zna otprilike što treba tražiti. Također, početna pozicija do sebe ima model strelice koja igraču signalizira da bi bilo dobro da se kreće određenim putem. Kada igrač odluči krenuti tim putem, dočekaju ga ključevi koji ga vode do prve tajne, ali ga također ga dočekaju i čudovišta s ciljem da nauči mehaniku njihova uništavanja. O svakoj pojedinoj funkcionalnosti igra brinu se različite komponente dodijeljene objektima u igri. Kako bi igrač imao što bolji osjećaj interakcije moramo odrediti iz koje će perspektive on igrati. Tako možemo odabrati prvo i treće lice. Odabrano treće lice zahtijeva da položaj kamere slijedi

glavnog lika u fokusu i da se kamera simultano pomiče s modelom lika. Obzirom da je teško vidjeti čudovišta iz perspektive iza igrača, pritiskom tipke C kamera se prebacuje na ptičju perspektivu te se može gledati odozgo kako je algoritam za traženje puta u grafu primijenjen. Glavni cilj igre igrač može postići kretanjem. Postoje dvije vrste kretanja: trčanje i hodanje. Nakon što smo definirali kretanje igrača, vrijeme je za početak avanture. Kako igrač istražuje svijet tako nailazi na područja u kojem se stvaraju čudovišta koja mu mogu nauditi ili čudovišta koja ga vode do cilja. Ta područja su okidači koji osluškuju ulazi li igrač u njihovu blizinu te ako je ta tvrdnja potvrdna stvaraju čudovišta. U slučaju da je to okidač koji stvara neprijateljska čudovišta, generirat će se nasumični broj između dva i devet te će toliko čudovišta u tom trenutku biti stvoreno. Svako čudovište na sebi nosi komponentu audio zvuka koji je također indikator da još uvijek negdje postoje čudovišta pa tako i opasnost. Od trenutka stvaranja čudovišta pokreće se vremenska kontrola, koji ima funkciju isključivanja okidača kako se igrač ne bi morao nositi prevelikim brojem neprijatelja odjednom. Čudovišta koja mu mogu nauditi imaju funkciju napada. I također ga slijede sve dok ih on ne uništi. Kretanje čudovišta je izvedeno primjenom algoritma A*. Kada god se crvena čudovišta nađu u blizini igrača, igraču se dodijeli određena šteta za „zdravlje“, naravno, igraču nije u interesu izgubiti svo zdravlje. Vizualna reprezentacija zdravlja implementirana je kao klizač koji se svakim napadom čudovišta ažurira i ovisno o postotku igračevog zdravlja mijenja boju od zelene, žute do crvene. Zelena boja reprezentira igraču sigurnost da još uvijek ima dovoljno zdravlja te nije u opasnosti. Zelena se boja mijenja u žutu na granici od pedeset posto ukupnog zdravlja. Žuta boja reprezentira upozorenje i mijenja se u crvenu na granici od dvadeset i pet posto ukupnog zdravlja. Crvena boja je krajnje upozorenje igraču da mora biti oprezan tokom igre. U slučaju da igrač izgubi svo zdravlje ponovno se inicira njegovo stvaranje na početnu poziciju te mu se vizualnom reprezentacijom daje do znanja da je umro. Igrač može također nauditi čudovištima pritiskom tipke za razmak u slučaju kada se nađe u blizini čudovišta. Zdravlje čudovišta implementirano je na identičan način kao i zdravlje igrača uz jednu razliku. Svako čudovište ima nasumično generiranu količinu zdravlja između vrijednosti 10 i 50. Uništavanjem neprijateljskog čudovišta, igrač kreće u daljnju avanturu i nailazi na predmete koji se mogu pokupiti. Ključevi su strateški raspoređeni kako bi prilikom prvog pristupa igri igrač sakupljanjem ključeva odmah pronašao prvu tajnu.

5 Razvoj videoigre

5.1 Razvojna okolina

Postupak razvoja sveo se na nekoliko faza. Istraživanje rada programskog alata Unity, testiranje najboljih algoritama traženja puta, modeliranje te implementacija. Svaki od ovih koraka proveden je na osobnom računalu sljedećih specifikacija: procesor AMD Ryzen 3 2300U s integriranom grafičkom karticom Radeon Vega Mobile GFX 2.00GHz sa 8.00 GB radne memorije. Operacijski sustav postavljen na računalu je MS Windows 10. Alpha verzija igre Mystery na računalu navedenih specifikacija vrti se sa stabilnih 100 do 110 slika u sekundi (engl. **frames per second**, skr. **FPS**). Za potrebe izrade igre korišten je alat Unity verzije Unity 2019.2.14f1, za potrebe modeliranja korišten je alat Blender verzije Blender 2.81.

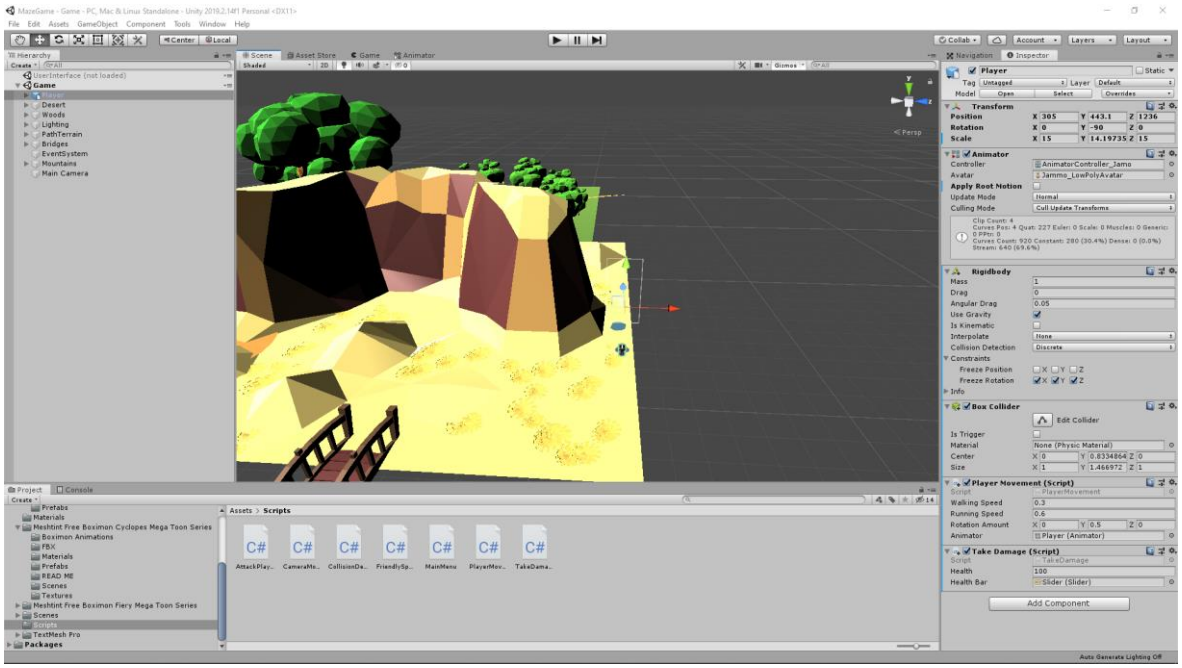
5.2 Alati

Izrada videoigre zahtjeva mnoga znanja pa tako i korištenje razne programske podrške koja omogućava početniku da lakše nauči, a iskusnom dizajneru da bude brži i efikasniji prilikom izrade igre. Radeći igru „*from scratch*“, odnosno od najosnovnijih „sastojaka“, može biti jako zahtjevno pa se stoga koristi programska podrška koja se brine o osnovnim stvarima za samu igru, koju dizajner zatim nadograđuje. Postoji nekoliko alata za izradu videoigara od kojih se najviše ističu Unity [https://unity3d.com/get-unity/download] i Unreal Engine [https://www.unrealengine.com/en-US/download]. Mnoge današnje igre su nastale korištenjem upravo jednog od ova dva alata. Svaki od njih ima svoje mane i prednosti.

5.2.1 Unity

Unity je programska podrška za izradu videoigara u dvodimenzionalnom ili trodimenzionalnom svijetu. Prvi puta je objavljen u lipnju 2005. godine na konferenciji tvrtke Apple. Danas ima podršku za izradu videoigara za više od 27

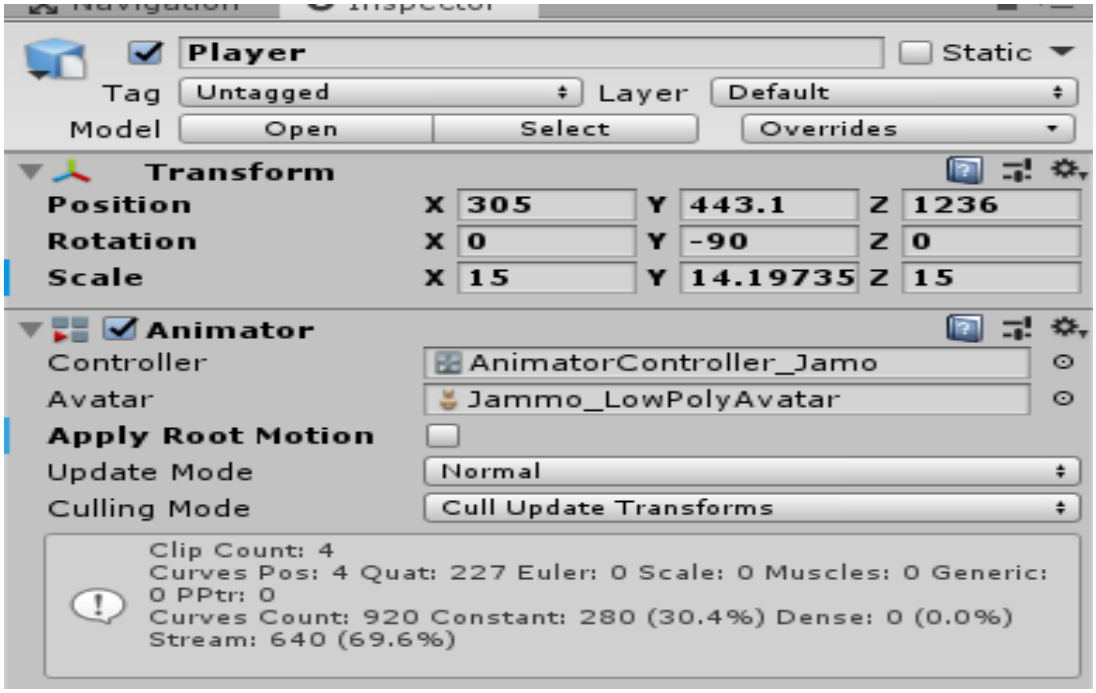
platformi, uključujući Android, PC igre, igre za konzole, igre u virtualnoj stvarnosti.



Slika 23 Izgled sučelja programskog alata Unity

Unity spaja umjetnički način izražavanja sa dizajnom baziranim na komponentama. Komponenta se primjenjuje na objekt u igri i definiramo ju kao svojstvo tog objekta. Komponente možemo napisati sami, a postoje i već prethodno izrađene komponente zbog čega je Unity vrlo dobar izbor, bilo za početnika u izradi videoigara, ili naprednijeg programera. Prethodno izrađene komponente brinu o osnovnim obilježjima koje bi objekt u igru trebao imati, od „**sudarača“** koji nam govore kada su se objekti sudarili, preko fizikalnih pojava

koje se događaju u stvarnom svijetu do komponenti vezanih za korisničko sučelje.



Slika 24 Izgled sučelja koje kontrolira komponente

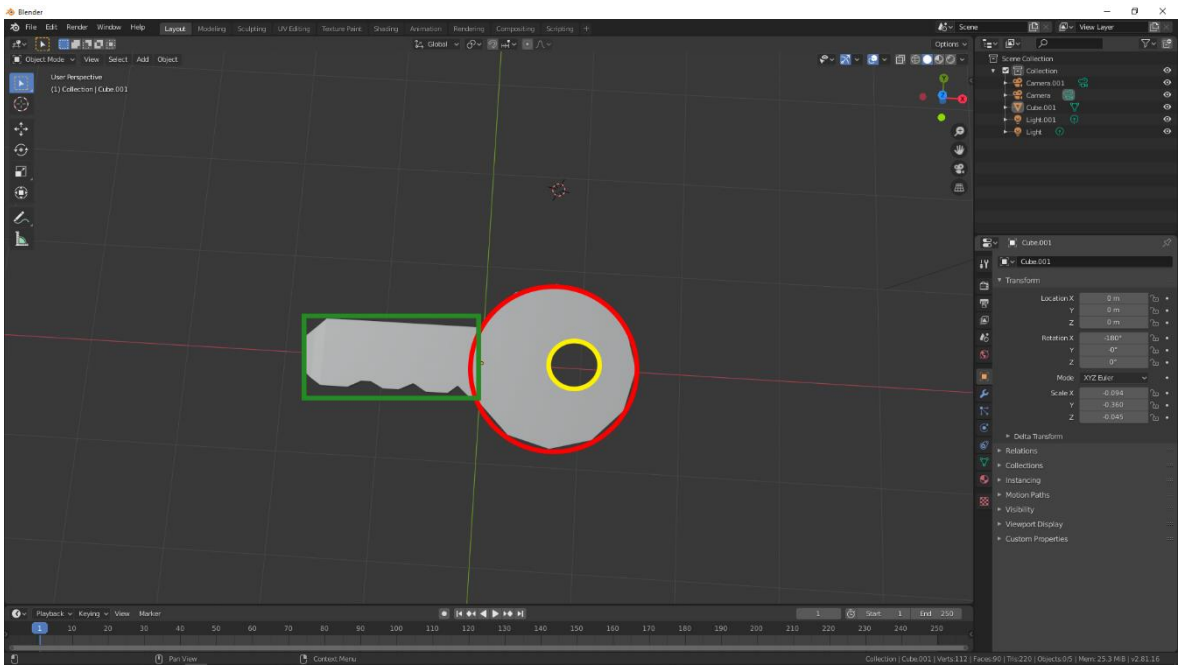
Komponente također možemo izrađivati sami tako da izradimo skriptu u programskom jeziku C# koji je objektno orijentiranog tipa pa možemo povezivati komponente međusobno kako bi utjecali na pojave u igri. Unity ima integriranu podršku za Microsfot Visual Studio u kojem se programira odnosno pišu skripte. Druga opcija programskog jezika jest Javascript. Unity također ima zbirku objekata,3d modela, animacija, tekstura, koje se prodaju u njihovoj trgovini gdje korisnici mogu također objavljivati svoje objekte. Za početnike u razvoju videoigara na raspolaganju je dokumentaciju Unityja (https://docs.unity3d.com).

5.2.3 Blender

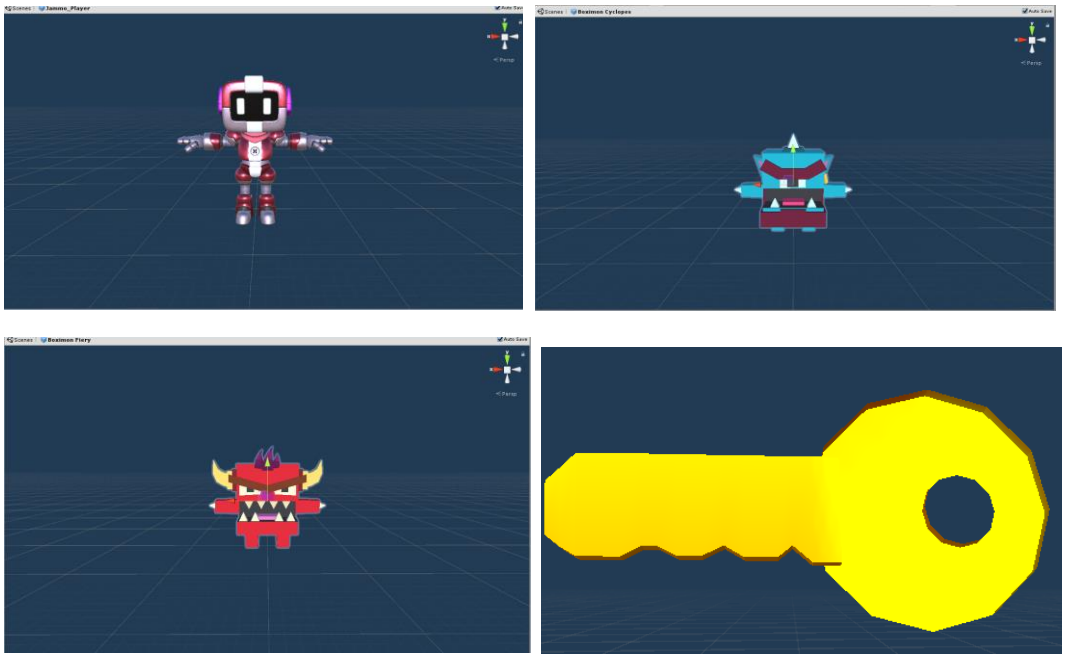
Blender [https://www.blender.org/download/] je besplatan profesionalni otvorenog koda alat za modeliranje trodimenzionalnih objekata, izradu računalno generiranih filmskih scena, vizualnih efekata, animacija. Blender podržava cjelovitost 3D protočne strukture, modeliranje, teksturiranje, animacija, simulacija, praćenje kretanja, uređivanje videa i kreiranje igara. Sučelje Blendera koristi OpenGL za izvršavanje operacija računalne grafike.

5.3 Izvedba modela

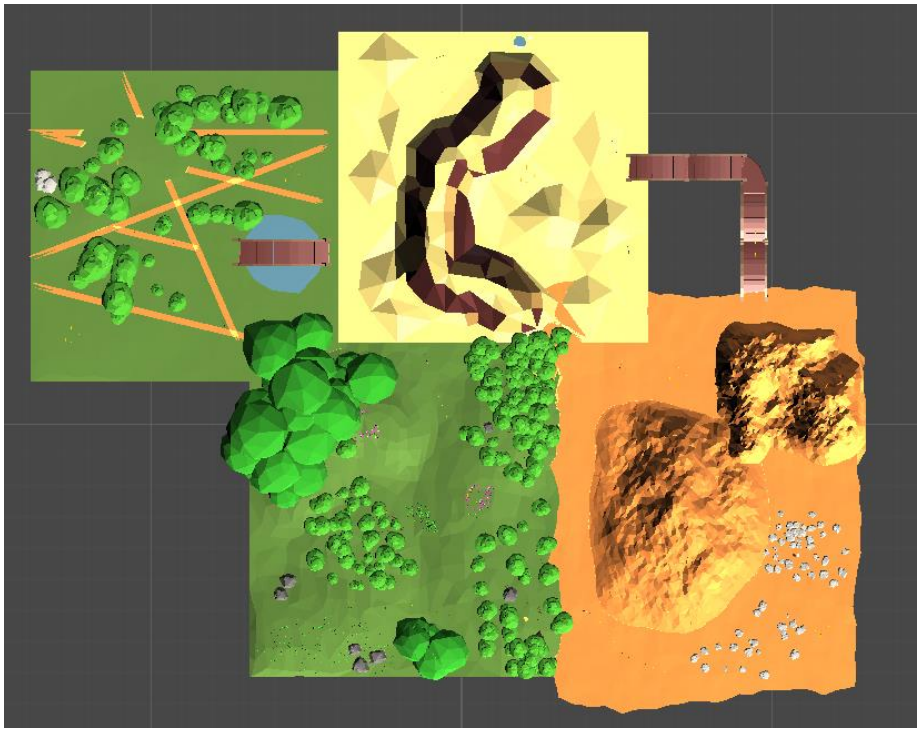
Za modeliranje je korišten alat Blender. Prvi susret s alatom može djelovati doista zbunjujuće no uz malo uloženog truda vrlo se brzo pohvataju konci izrade 3D modela. Misija na kojoj igrač skuplja ključeve jest otkriti, otključati svoje podrijetlo i upravo je zbog te simbolike otključavanja odabran ključ koji će igraču dati neki dodatan izazov u igri. Za referencu ključa korištena je dvodimenzionalna slika ključa pomoću koje su određene proporcije. Sam model ključa vrlo je jednostavan, sastoji se od jednog valjka i nekoliko kvadara koje su manipulacijom dovedene da izgledaju kao ključ. Na slici crvenom bojom je označen valjak, žutom bojom je označen manji valjak kojim smo, primijenivši operaciju razlike sa crvenim valjkom, dobili rupu na ključu. Zelenom bojom označene su svi kvadri. Svi modeli, korišteni prilikom izrade igre Mystery, dostupni su na https://assetstore.unity.com isključujući model ključeva koji je rad samostalnog modeliranja.



Slika 25 Seciranje modela na dijelove



Slika 26 Izgled modela robota, plavog čudovišta, crvenog čudovišta te ključa



Slika 27 Izgled svijeta, 4 različita područja sa specifičnim skrivenim pojavama

5.4Izvedba interakcija

Na početku igre igraču se prikazuje „*tutorial*“ koji mu objašnjava kako se može kretati. Igrač mijenja poruke tog tutoriala pritiskom tipke „*Tab*“ sve dok ne dođe do kraja istog. Kako se igrač kreće u svijetu tako se pokreće i kamera. Dvije su perspektive na kojima kamera može biti pozicionirana: ptičja perspektiva i perspektiva gdje se kamera nalazi iza leđa igrača. Možemo kontrolirati „offset“ ili udaljenost kamere od igrača, što se može protumačiti kao field of view, a simultanu rotaciju kamere, s modelom robota, postižemo funkcijom *lookAt* koja kao parametar uzima položaj prema kojem bi se kamera trebala okrenuti te određuje kut kako bi kamera gledala u leđa igraču.

```
public class CameraMovement : MonoBehaviour
{
    public Transform player;
    public Vector3 offset;
    public float cameraDistance;
    public bool cameraUP = false;

    void LateUpdate()
    {
        if (Input.GetKeyDown(KeyCode.C)) {
            cameraUP = !cameraUP;
        }
        if (cameraUP) {
            transform.position = player.transform.position - player.transform.up*(-1)
* cameraDistance;
            transform.LookAt(player.transform.position);
            transform.position = new Vector3(transform.position.x,
transform.position.y + 100, transform.position.z);
        }
        else {
            transform.position = player.transform.position - player.transform.forward
* cameraDistance;
            transform.LookAt(player.transform.position);
            transform.position = new Vector3(transform.position.x + offset.x,
transform.position.y + offset.y, transform.position.z + offset.z);
        }
    }
}
```

Slika 28 Skripta koja brine za kretanje kamere

Kretanje igrača, kroz virtualni svijet, izvedeno je od dva dijela. Jedan dio jest animacija koja se odvija u pozadini za svo vrijeme dok traje određeno kretanje te drugi dio koji je skripta koja brine o pozicioniranju i rotaciji modela u svijetu. Skripta ili kontroler kretanja vrlo je jednostavna: ovisno o pritisku tipke pomiče i rotira model robota kako bi igrač dobio dojam da on reagira na njegov unos. Modelu robota dodijeljena je komponenta „RigidBody“ koja pruža Unity, ona implementira

fizikalne pojave stvarnog objekta. Također kako bi se mogao sudarati s predmetima robotu je dodijeljena komponenta „CapsuleCollider“ , bez koje bi model slobodno padao u virtualnom svijetu.

```
public class PlayerMovement : MonoBehaviour
{
    float speed;
    public float walkingSpeed;
    public float runningSpeed;
    Rigidbody rbody;
    public Vector3 rotationAmount = new Vector3(0, 0.5f, 0);

    public Animator animator;

    // Start is called before the first frame update
    void Start()
    {
        rbody = GetComponent<Rigidbody>();
        speed = walkingSpeed;
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
        {
            transform.Translate(Vector3.forward * speed);
            animator.SetFloat("speed",3f);
        }
        if (Input.GetKeyUp(KeyCode.W) || Input.GetKeyUp(KeyCode.UpArrow))
        {
            animator.SetFloat("speed", 0f);
        }

        else if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))
        {
            this.transform.Rotate(rotationAmount*-1, Space.World);
        }
        else if (Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow))
        {
            this.transform.Rotate(rotationAmount, Space.World);
        }
        else if (Input.GetKey(KeyCode.LeftShift))
        {
            speed = runningSpeed;
            animator.SetFloat("speed", 5f);

        }
        else if (Input.GetKeyUp(KeyCode.LeftShift))
        {
            speed = walkingSpeed;
            animator.SetFloat("speed", 0f);
        }
    }
}
```

Slika 29 Skripta koja brine za kretanje igrača

Područje stvaranja čudovišta ima funkciju gdje se nasumično stvara broj čudovišta između dva i devet i svakome od njih se dodijeli nasumična vrijednost njihovog „zdravlja“. Igrač može ta čudovišta uništavati pritiskom tipke *Space.* Svako

čudovište na sebi ima „*Audio Source*“ koji se vrti u petlji te tako implicira igraču da je čudovište još živo i da ga napada.

```
public class EnemySpawn : MonoBehaviour
{

    public GameObject GO_Enemy;
    public GameObject player;
    GameObject spawnedMonster;
    Vector3 spawnPos;
    List<GameObject> spawnedMonsters = new List<GameObject>();

    private void OnTriggerEnter(Collider other)
    {
        if (!GetComponent<TimerForSpawning>().spawned)
        {
            if (other.gameObject.name.Equals("Player"))
            {
                //Debug.Log("TRIGGER");
                //Debug.Log(spawnPos);
                for (int i = 0; i < Random.Range(3, 9); i++)
                {
                    spawnPos = transform.position + Vector3.forward * Random.Range(-
50f, 50f);
                    spawnPos += Vector3.right * Random.Range(-50f, 50f);
                    spawnedMonster = Instantiate(GO_Enemy, spawnPos,
Quaternion.identity);
                    spawnedMonster.transform.localScale += new Vector3(5, 5, 5);
                    spawnedMonster.transform.Rotate(0, 90, 0);
                    spawnedMonsters.Add(spawnedMonster);
                }
                addComponents();
            }
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (GetComponent<TimerForSpawning>().spawned == false) {
            GetComponent<TimerForSpawning>().spawned = true;
        }
    }

    private void addComponents()
    {
        foreach (GameObject spawnedMonster in spawnedMonsters) {

            spawnedMonster.GetComponent<AttackPlayer>().player = player;
            spawnedMonster.GetComponent<NavMeshAgent>().speed = Random.Range(10f,
30f);
        }
    }
}
```

Slika 30 Skripta koja brine za stvaranje neprijateljskih čudovišta

Čudovište može nauditi igraču funkcijom „*sendDamage“* koja se poziva kad se igrač nađe u blizini čudovišta. Igraču se smanjuje „zdravlje“ svakim pozivom funkcije „sendDamage“ i mijenja se vrijednost klizača koji ima funkciju prikaza „zdravlja“.

```
private void Update()
    {
        agent.destination = player.transform.position;
        animator.SetBool("Run Forward", true);
        Vector3 distance = player.transform.position - transform.position;

        if (distance.magnitude < 10)
        {
            sendDamage(0.2f);
        }
    }
    void sendDamage(float damage) {
        HealthBar take = player.GetComponent<HealthBar>();
        take.takeDamage(damage);
    }
```

Slika 31 Dio koda kojim čudovište oduzima "zdravlje" igraču

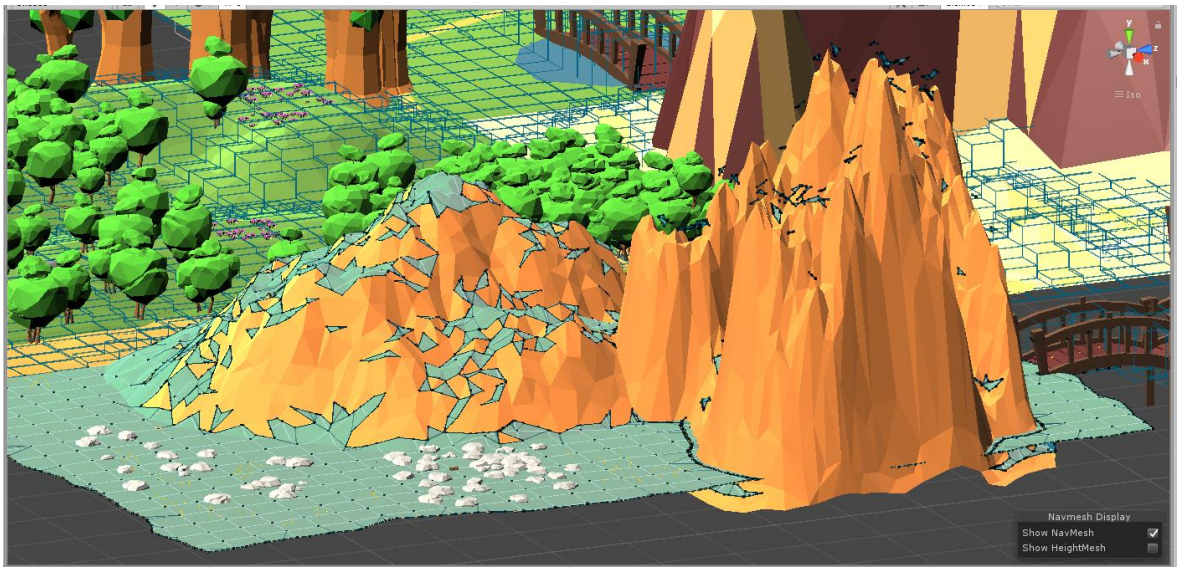
Kako se igrač kreće u svijetu tako nailazi i na ključeve koji su predmeti koji se mogu skupiti. Svaki ključ na sebi ima okidač. U slučaju da igrač uđe u polje okidača, povećava se na korisničkom sučelju broj skupljenih predmeta zatim se pokreće zvučni efekt i na kraju se taj objekt, u čiji prostor okidača je igrač ušao, uništava.

```
public class ItemCollected : MonoBehaviour
{

    private void OnTriggerEnter(Collider other)
    {
        if (other.transform.name.Equals("Player")) {
            if (gameObject.tag.Equals("keys"))
            {
                other.GetComponent<Collectibles>().updateNumberOfCollectedCoins(1);

GameObject.FindGameObjectWithTag("MainCamera").GetComponent<AudioSource>().Play(0);
                Destroy(gameObject);
            }
        }
    }

}
```

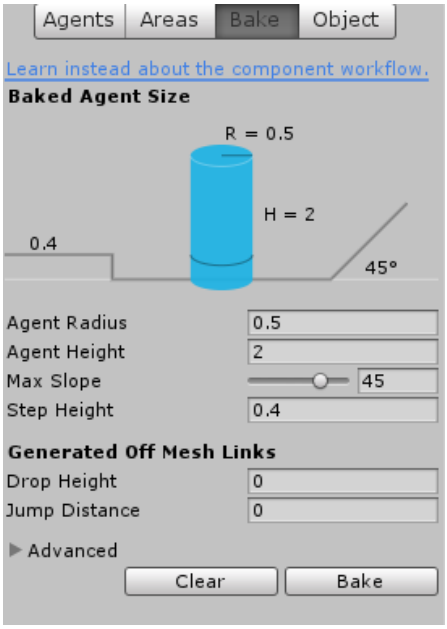
Slika 32 Skripta koja brine o skupljanju ključeva

5.5 Primjena algoritma za traženje puta

Algoritam koji se primjenjuje u ovom slučaju jest algoritam A*, zato što u većini slučajeva pronalazi optimalan put i k tome je još brz, uz manji broj izvedenih operacija što znači brže izvođenje i manje opterećenje procesora. Model crvenog čudovišta na sebi će imati primijenjen algoritam, njegovo krajnje odredište bit će igrač. Tom primjenom čudovište se pokušava približiti igraču na mapi, odnosno slijedi ga. Kako bi se čudovište moglo kretati trebalo je prilagoditi teren za kretanje, odnosno treba izgraditi neku vrstu mreže pomoću koje će čudovište i algoritam dobivati informacije o ciljnoj poziciji odnosno igraču. Unity pruža alat kojim se upravo izgrađuje mreža za kretanje. Definiraju se mjesta na kojima agent može stajati i kretati se, tzv. „walkable areas“. Područja kojima se agenti mogu kretati izgrađuju se na način da se provode testovi kojima se ispituje na kojem položaju agent može stajati, odnosno gdje se događa kolizija sa mapom. Tada se ta područja postavlja na mapu, odnosno prilagođavaju se geometriji mape. Tako izgrađeno područje za kretanje naziva se NavMesh. NavMesh sprema ta područja kao konveksne poligone iz razloga što znamo da unutar jednog poligona nema nikakvih prepreka između bilo koje dvije točke. Uz veličinu i položaj poligona, odnosno njegove rubove NavMesh također sprema informaciju o susjednim poligonima i time je efektivno generiran graf sa stanjima koji reprezentiraju položaj u mapi. Postupak izgradnje navigacijske mreže vrlo je jednostavan. Odabere se područje za koje se želi napraviti mreža te se sa desne strane nalazi se izbornik Inspector te pokraj njega Navigation. Odabirom tipke Navigation otvara se izbornik Object, a pokraj njega, s lijeve strane nalazi se izbornik Bake. Odabirom izbornika Bake otvara se sučelje u kojem možemo kontrolirati varijable bitne za navigacijsku mrežu. Na mapi postoje prepreke za koje želimo da čudovišta izbjegavaju, odnosno da znaju kako tu postoji prepreka te da se simulira zaobilaženje prepreke. Kako bismo ostvarili taj efekt na svaku prepreku dodajemo komponentu NavMeshObstacle u obliku kutije ili cilindra ovisno o potrebama te prilagođavamo veličinu kako bi odgovarala našoj prepreci.



Slika 33 Izgled jednog dijela mape nakon izgradnje navigacijske mreže



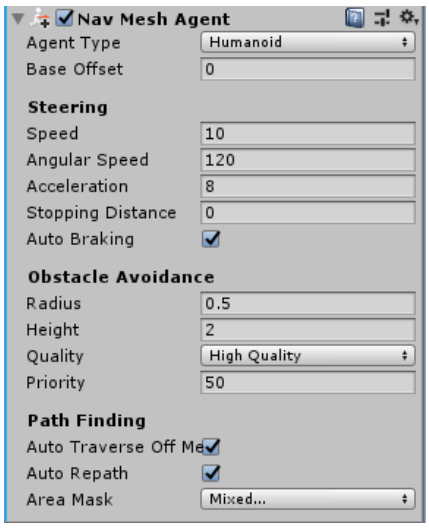
Slika 34 Izbornik Bake

U izborniku Bake ono što je bitnije za izgradnju same navigacijske mreže jest radijus i visina agenta koji će šetati po toj navigacijskoj mreži. Visina je bitna u slučaju da agent prolazi ispod neke prepreke pa ako će biti previsok taj put će se zanemariti, a širina naravno zbog prepreka koje dolaze sa strane. Kako bi se zapravo naš agent, odnosno čudovište moglo kretati po mapi na njega moramo primijeniti komponentu NavMeshAgent. Kako bismo čudovištu dodijelili komponentu NaveMeshAgent klikom na model čudovišta s desne strane otvara se Inspector gdje klikom na „add component“ odaberemo komponentu NavMeshAgent. Ovdje možemo kontrolirati brzinu, akceleraciju kretanja i razne

druge parametre. No, samo dodavanje te komponente nije dovoljno da se agent zapravo kreće po mapi. Za to nam je još potrebna jedna skripta kojom ćemo postavljati cilj agenta: doći do igrača. Kada primijenimo tu skriptu na naš model čudovišta implementirali smo algoritam za traženje puta u grafu i čudovište će u svakom trenutku slijediti igrača po mapi najboljim mogućim putem.

```
public class AttackPlayer : MonoBehaviour
{
    NavMeshAgent agent;
    public Animator animator;
    public GameObject player;

    private void Start()
    {
        agent = GetComponent<NavMeshAgent>();
        player = GameObject.FindGameObjectWithTag("Player");
    }

    private void Update()
    {
        agent.destination = player.transform.position;
        animator.SetBool("Run Forward", true);
        Vector3 distance = player.transform.position - transform.position;

        if (distance.magnitude < 10)
        {
            sendDamage(0.2f);
        }
    }
    void sendDamage(float damage) {
        HealthBar take = player.GetComponent<HealthBar>();
        take.takeDamage(damage);
    }
}
```

Slika 35 Skripta odgovorna za kretanje i napad igrača



Slika 36 Inspector komponente NavMeshAgent

6 Rezultati

Kao rezultat ovog rada nastala je igra Mystery. Scenarij igranja opisan je kroz slike u nastavku. Igra počinje stvaranjem igrača na početnoj poziciji i prikazuje mu se kratke upute koje mu objašnjavaju koji je njegov cilj i kako bi taj cilj trebao postići. U gornjem lijevom kutu vidimo broj sakupljenih ključeva te broj ključeva koje još treba skupiti, dok u desnom kutu vidimo „zdravlje“ igrača

Slika 37 Početno stanje igre

Igrač može mijenjati perspektivu kamere pritiskom na tipku *C* kako bi dobio dojam rada algoritma i bolji pogled na mapu.

Slika 38 Pogled odozgo

Kretanjem nailazi na ključeve koje može skupiti.

Slika 39 Igrač je pokupio ključ

Na mapi se nalaze strelice koje ga upućuju u kojem smjeru bi se trebao kretati kako bi pronašao tajne, kroz cijeli put ga prate neprijateljska čudovišta koja mu otežavaju pronalazak i sakupljanje ključeva.

Slika 40 Igrač je stigao do strelice pri čemu ga napadaju čudovišta

Kada igrač konačno dođe do područja koje će mu otkriti tajnu pojavljuje se prijateljsko, plavo, čudovište koje mu govori kako je pronašao tajnu. Postoje takva četiri prijateljska čudovišta, po jedan na svakom od područja u virtualnom svijetu.

Slika 41 Svako područje ima jednu nespecifičnu stvar

7 Zaključak

Trodimenzionalne igre i igre općenito su u današnjem društvu vrlo popularne. Omogućavaju bijeg od stvarnosti, zabavu, ali i učenje. Gledajući razvoj videoigara kroz povijest shvatiti ćemo da igre danas imaju dimenziju umjetnosti. Vrlo su usko povezane s filmskom industrijom no imaju koncept koji ih odvaja od filmske industrije, a to je uključenost osobe u proces igre. U razvoju se, općenito, isprepliću znanja iz tehničkog svijeta te znanja o umjetničkim komponentama igre. U okviru ovog završnog rada razvijena je igra naziva Mystery o robotu koji se nalazi, u njemu nepoznatom svijetu te nema ideju kako se našao na njemu i kretanjem kroz svijet pokušava otkriti zašto je ovdje dok nailazi na prepreke. U razvoju programskog rješenja korišten je programski alat Blender za modeliranje trodimenzionalnih modela i programski alat Unity, te programski jezik C# koji služi za pisanje komponenti koje se dodaju objektu. Poznavanje algoritama za traženje puta u grafu također igra važnu ulogu u odlučivanju koji je algoritam najbolji, i zašto, za primjenu u razvoju vlastite igre. Analizom algoritama BFS, najbolji prvi i A* odluka je pala na algoritam A* zbog brzine izvođenja i manje količine memorije koja mu je potrebna za njegovo izvršavanje.

8 Literatura

[1] Blender manual - < https://docs.blender.org/manual/en/latest/ > (prosinac 2019.)

[2] Ernest Adams,(2010.),“Fundamentals of Game Design“, drugo izdanje, Pearson Education

[3] Comparative Analysis of Path Finding Algorithms - < http://www.iosrjournals.org/iosr-jce/papers/Vol20-issue5/Version-1/E2005013845.pdf > (prosinac 2019.)

[4] Diagram editor - < https://www.diagrameditor.com/ > (prosinac 2019.)

[5] Introduction to the A* Algorithm - < https://www.redblobgames.com/pathfinding/a-star/introduction.html > (prosinac 2019.)

[6] J. Šnajder, B. Dalbelo-Bašić,(2017.), prezentacije o heurističkom pretraživanju i pretraživanju prostora stanja s FER-a

[7] Steven L. Kent, (2001.),“The Ultimate History of Video Games“

[8] Unity manual - < https://docs.unity3d.com/Manual/index.html > (prosinac 2019.)

[9] Vizualizacija traženja puta u grafu – < https://qiao.github.io/PathFinding.js/visual/ > (prosinac 2019.)

Naslov :

Primjena algoritama traženja puta u grafu pri razvoju jednostavne 3D videoigre

Sažetak :

Videoigre mogu biti vrlo koristan alat za istraživanje nepoznatog i oponašanja fizikalnog svijeta, ali također izvrstan način za vizualizaciju algoritama. Igra naziva Mystery razvijena u okviru ovog završnog rada, tako u sebi ima ugrađen algoritam za traženje puta u grafu, primijenjen na računalno generiranim likovima koji slijede igrača u virtualnom trodimenzionalnom svijetu. Na temelju analize tri algoritma za traženje puta u grafu (BFS, najbolji prvi, A*), koja uključuje vrijeme izvršavanja i broj izvršenih operacija na pet različitih mapa, za primjenu u igri odabran je algoritam A*. Opisan je postupak izrade trodimenzionalnih modela u alatu Blender te postupak dizajna i izrade igre.

Ključne riječi :

Unity, Blender, videoigra, traženje puta, A*,BFS, pohlepni prvi

Title :

Application of pathfinding algotihms in a simple 3D videogame development

Abstract :

Virtual games can be very useful tool for exploring the unknown and modeling the real physical world, but also a great way of visualizing the inner workings of algorithms. The game, named of Mystery, includes a pathfinding applied to computer generated characters following the player inside the virtual world. Based on an analysis of the three pathfinding algorithms, including BFS, first best and A*, and their runtime and number of operations, A* has been selected to be applied in the game implementation. The process of creating three-dimensional models in Blender tool is described, as well as the overall process of the game design and implementation.

Key words :

Unity, Blender, videogame, pathfinding, A*,BFS, first best