UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Stefan Janjić

Grafovske gramatike i primena na proceduralno generisanje misija u video igrama

master rad

Beograd, 2018.

Mentor:

dr Filip Marić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Predrag Janičić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Milan Banković, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Apstrakt

Proceduralno generisanje misija je jedan od vidova automatskog kreiranja sadržaja u video-igrama. Mnoga rešenja koriste grafovkse gramatike kao osnovu tehnike generisanja. Korišćenjem skupa pravila grafovske gramatike, kroz više koraka prezapisivanja, vrše se transformacije nad definicijom misije. Opisujemo metodu prezapisivanja grafova i generator misija, implementiranog u vidu dodataka *Unity* sistemu. Implementirana procedura prezapisivanja zasnovana je na metodi dvostrukog potiska, dok je za pronalazak izomorfizma u podgrafu upotrebljen algoritam *VF2*. Prilikom razvoja dodatka iskorišćena je ugrađena mogućnost pisanja dodataka u okviru *Unity* sistema. Dodatak daje mogućnost definisanja pravila gramatike i parametara prezapisivanja u cilju generisanja grafa misije za video-igru. Dalji razvoj bi mogao da uključi integraciju sa sistemima za generisanje opisa prostora i geometrije nivoa video-igre.

Sardžaj

1

Uvod

Grafovske gramatike predstavljaju skupove pravila za transformaciju grafova. Koriste se za rešavanje problema kao što su prepoznavanje i generisanje obrazaca, procesiranje slika, izračunavanje algebarskih izraza i mnogih drugih. Njihova interesantna primena je u programskim paketima koji se koriste za proceduralno generisanje sadržaja, što ima za cilj da olakša ili u potpunosti automatizuje kreiranje različite vrste sadržaja. Metode proceduralnog generisanja se često koriste za kreiranje sadržaja u video-igrama. Ne tako veliki broj igara koristi ovakve metode, iako one postoje već mnogo godina. Svakako, popularnost se povećava s vremenom, dok se nove metode paralelno razvijaju i unapređuju.

Razlozi za korišćenje generisanog sadržaja menjali su se kroz istoriju, a zajedničko im je pokušaj poboljšavanja nekog vida performansi. Prvobitno su bile u pitanju performanse računara. Osamdesetih godina, pojavom igara sa grafikom, računari su imali veliko memorijsko ograničenje. Iz tog razloga, sadržaj je morao biti generisan tokom izvršavanja igre korišćenjem različitih algoritama. U suprotnom, nije bilo dovoljno prostora za njegovo čuvanje. Kasnijim napretkom tehnologije, memorijski problemi su bili rešeni, ali je proceduralno generisanje našlo drugačiju upotrebu. Kod modernih igara, sadržaj se generiše kako bi se pobaljšale performanse procesa razvoja. Određeni skup sadržaja se može automatskim putem brzo generisati, što utiče na efikasnost razvoja i omogućuje razvijaocima da pruže više pažnje drugim aspektima razvoja video-igre.

Danas razvoj igara zahteva proizvodnju velike količine sadržaja različite prirode. Od geometrije kojom se ispunjava prostor igre, pa sve do njihove logičke strukture. Kako bi prostori izgledali realistično, nivoi se često popunjavaju različitim vidovima vegetacije. Kreiranje i raspoređivanje biljaka u prostoru može biti mukotrpan, dug i dosadan posao. *SpeedTree* [7] je programski paket koji to rešava primenom različitih metoda proceduralnog generisanja. Primer generisanja logičke strukture je kreiranje misija, što će i biti glavna tema ovog teksta. Misija predstavlja skup zadataka i prepreka koje igrač mora da reši, za šta potencijalno može postojati više načina, u cilju njenog uspešnog završavanja.

Proceduralno generisanje ima sve bitniju ulogu u razvojnom procesu video-igara, ne samo kad je u pitanju povećanje efikasnosti tokom razvoja već i u kreativnom procesu. Već spomenuti *SpeedTree* i drugi alati, daju mogućnost da dizajner dođe do možda ne tako očiglednih rezultata tokom dizajniranja. Širok spektar sadržaja koji može biti kreiran ovakvim metodama dovodi do toga da se cele igre mogu zasnovati da konceptima proceduralnog generisanja. Jedan primer video-igre koja

proceduralno generisanje koristi u ovako velikom obimu je *No Man's Sky*. Pored toga, ovu igru možemo uzeti i za primer koji pokazuje da mali studio može da napravi proizvod sa velikom količinom sadržaja i plasira ga na tržište.

Cilj rada je da predstavi jednu od tehnika za proceduralno generisanje misija u video-igrama korišćenjem metode dvostrukog potiska za definisanje grafovskih gramatika i njenih pravila. Pored toga, predstavićemo algoritam za pronalaženje izomorfnog podgrafa za date grafove, kao bitan deo primene pravila prezapisivanja. Misije u video-igrama posmatramo kao strukturu koja definiše ciljeve koje igrač mora da ispuni u zavisnosti od postavljenih uslova. Grafovi će nam omogućiti da intuitivno predstavljamo strukuru misije.

Biće reči i o implementaciji rešenja za proceduralno generisanje misija u vidu dodatka za Unity sistem. Dodatak treba da omogući dizajnerima da definišu početni graf i pravila gramatike čijom će se primenom u konačnom broju koraka dobiti struktura misije, ponovo u vidu grafa. Ovakav alat treba da služi kao pomoć tokom procesa dizajniranja misija.

2

Grafovi

Grafovi, kao jedna od osnovnih struktura koja će biti korišćena, imaju veoma široku primenu u računarstvu. Igraju veliku ulogu u metodama proceduralnog generisanja, ne samo kod generisanja misija već i u drugim slučajevima. Jedan od takvih primera je generisanje prostora za igranje u video-igrama. Ovde ćemo opisati i definisati osnovne pojmove koji će nam biti od važnosti u narednim izlaganjima.

2.1 Osnovne definicije

Definicija 1 *Usmerenig graf* $G = (V, E)$ se sastoji od konačnog skupa čvorova V i konačnog skupa grana E, pri čemu grane iz E predstavljaju uređene parove (a, b) čvorova iz V.

Konačni grafovi se predstavljaju dijagramima u kojima su čvorovi predstavljeni kružićima, a grane koje spajaju dva čvora, strelicama koje povezuju odgovarajuće kružiće. Strelicama je određen smer shodno definiciji uređenog para koji se nalazi u skupu E. Ovako definisane grane nazivaju se *usmerene grane* i grafovi sa takvim granama *usmereni grafovi*. U *usmerenom grafu* za svaka dva čvora a i b u skupu E mogu postojati uređeni parovi - grane *(a,b)* i *(b,a)*. U većini slučajeva ako želimo da *usmereni graf* razmatramo kao *neusmereni* graf, dovoljno je da za svaku granu *(a,b)* koja pripada skupu E dodamo i granu *(b,a)* u skup. Pored toga neusmereni graf možemo definisati i na sledeći način.

Definicija 2 *Neusmereni* graf $G = (V, E)$ se sastoji od konačnog skupa čvorova V i kolekcije grana E, dvoelementnih podskupova skupa V. Elementi a i b skupa V su povezani pomoću grane $\{a, b\}$ ako $\{a, b\} \in E$.

Navodimo definicje susednosti čvorova kod usmerenih i neusmerenih grafova.

Definicija 3 Ako je par *(a,b)* grana usmerenog grafa *G = (V,E)*, takav da je a početni čvor i b završni čvor, tada se za čvorove a i b kaže da su *incidentni* grani *(a,b)*. Čvor a je susedan ka čvoru b i čvor b je susedan ka čvoru a.

Na sličan način se definiše susednost čvorova kada su u pitanju *neusmereni grafovi*.

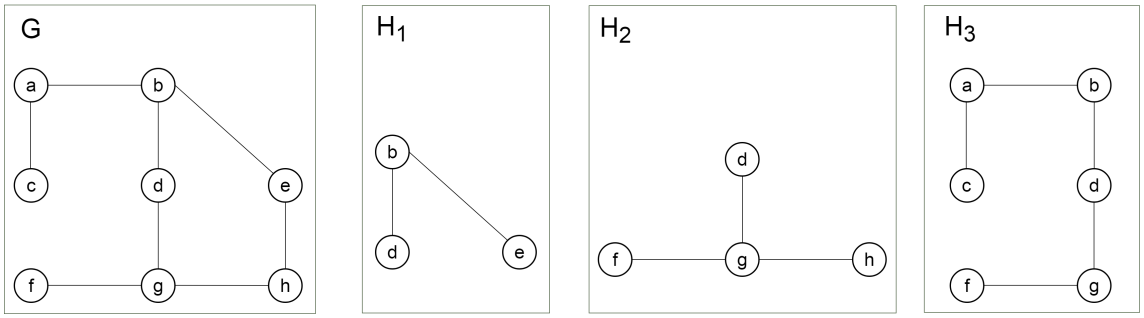
Definicija 4 Ako skup $\{a, b\}$ predstavlja granu grafa G, tada se čvorovi a i b nazivaju *krajnjim tačkama* grane $\{a, b\}$. Za granu $\{a, b\}$ se takođe kaže da je incidentna čvorovima a i b. Dva čvora su *susedna* ako predstavljaju krajnje tačke iste grane, ili ekvivalentno tome, ako su incidentni istoj grani.

Definicija 5 Ulazni stepen čvora $v \in V$, u oznaci $deg^+(v)$, predstavlja broj grana kojma je v završni čvor. Ako je $deg^+(v) = 0$, tada se čvor v naziva *izvorom*.

Definicija 6 Izlazni stepen čvora $v \in V$, u oznaci $deg^-(v)$, predstavlja broj grana kojma je v početni čvor. Ako je $deg^-(v) = 0$, tada se čvor v naziva *ponorom*.

Definicije *ulaznog i izlaznog* stepena, kao i naredna definicija *podgrafa*, biće nam od velikog značaja prilikom definisanja i implementacije algoritma za grafovski izomorfizam.

Definicija 7 Usmereni graf $G' = (V', E')$ je *usmereni podgraf* usmerenog grafa $G = (V, E)$, u oznaci $G' \preceq G$ ako važi $V' \subseteq V$ i $E' \subseteq E$.



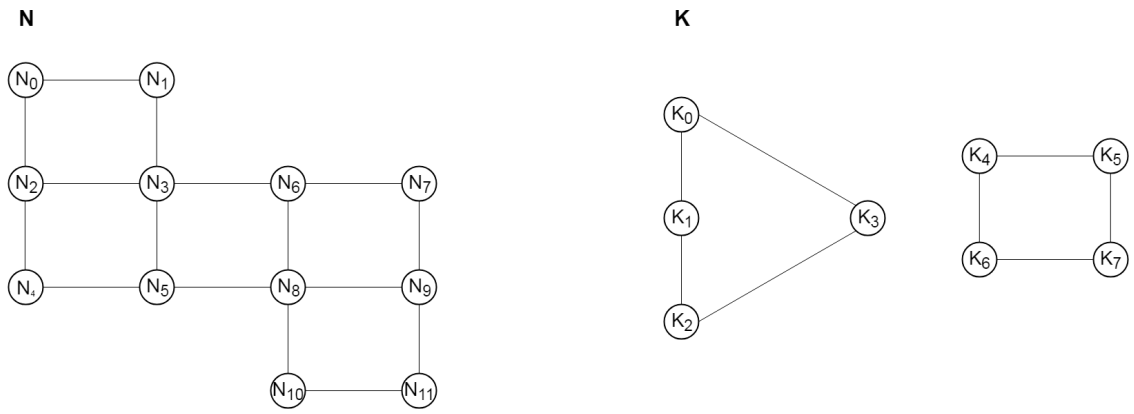
Slika 2.1: Grafovi H_1, H_2, H_3 predstavlaju različite podgrafe grafa G.

Navedena definicja nam govori da svaki čvor grafa G' je i čvor grafa G i svaka usmerena grana grafa G' je usmerena grana i grafa G.

Posmatraćemo grafove koji nemaju izolovane čvorove ili grupe povezanih čvorova gde su grupe međusobno izolovane. Izolovanost čvora definisaćemo pomoću stepena čvora.

Definicija 8 Čvor kojem su $deg^-(v) = 0$ i $deg^+(v) = 0$ naziva se *izolovanim*.

Pored izolovanih čvorova mogu postojati cele grupe izolovanih, ali međusobno povezanih čvorova, koje se obično nazivaju *ostrva*. Ipak, mi ćemo u obzir uzimati samo *povezane grafove*. Iz tog razloga, prvo definišemo *put* u neusmerenom i usmerenom grafu. Nakon toga, povezanost u neusmerenom grafu na osnovu koje ćemo definisati *nosećи* graf i *povezan* usmereni graf.



Slika 2.2: Primeri povezanosti kod neusmerenih grafova. Povezani graf N (levo) i nepovezani graf K (desno).

Definicija 9 Neka je $G = (V, E)$ neusmereni graf sa čvorovima $v_0, v_k \in V$. *Put* dužine k od čvora v_0 do čvora v_k predstavlja sekvencu čvorova $v_0v_1v_2, ..., v_{k-1}v_k$, takvu da čvorovi $v_0, v_1, ..., v_k$ pripadaju V i grana $\{v_i, v_{i+1}\}$ pripada E za svako i iz $\{0, 1, ..., k-1\}$.

Definicija 10 *Usmereni put* od čvora a do čvora b je opisan kao sekvenca čvorova $v_0v_1v_2...v_n$, gde je $a = v_0$, $b = v_n$, a (v_i, v_{i+1}) je usmerena grana koja pripada E za svako i iz $\{0, 1, ..., k-1\}$. *Dužina* puta predstavlja broj usmerenih grana u putu.

Definicija 11 Neusmereni graf $G = (V, E)$ je *povezan* ako postoji *put* između bilo koja dva različita čvora grafa G.

Za usmereni graf G prvo opisujemo neusmereni graf G^n, takav da svaka usmerena grana grafa G prelazi u neusmerenu granu grafa G^n.

Definicija 12 Za usmereni graf $G = (V, E)$, neka je $G' = (V, E')$, gde je $E' = E - \{(v, v)|v \in V\}$, tj. G' nastaje od grafa G uklanjanjem petlji. Neka je $E^n = \{\{a, b\}|(a, b) \in E' \vee (b, a) \in E'\}$. Tada neusmereni graf $G^n = (V, E^n)$ nazivamo noseći graf grafa G.

Neformalno, skup grana E^n je definisan kao $\{a, b\} \in E^n$, za različite čvorove a i b, ako i samo ako $(a, b) \in G$ ili $(b, a) \in G$.

Definicija 13 Usmereni graf $G = (V, E)$ je povezan ako je njegov noseći graf povezan. Usmereni graf je *jako povezan* ako za svaki par čvorova $a, b \in V$ postoji usmereni put od čvora a do čvora b.

2.2 Izomorfizam grafova

Jedan od izazova koji se pojavljuje prilikom generisanja misija pomoću grafovskih gramatika jeste pronalaženje izomorfnih podgrafa na osnovu datog pravila gramatike. Razlog za to u našem slučaju je primena istog pravila prezapisivanja na pronađeni podgraf. O pravilima i gramatikama će biti više reči u daljem tekstu, dok ćemo se sada baviti definisanjem grafovskog izomorfizma i opisivanjem algoritma za njegovo pronalaženje.

2.2.1 Definicije i problem izomorfizma podgrafa

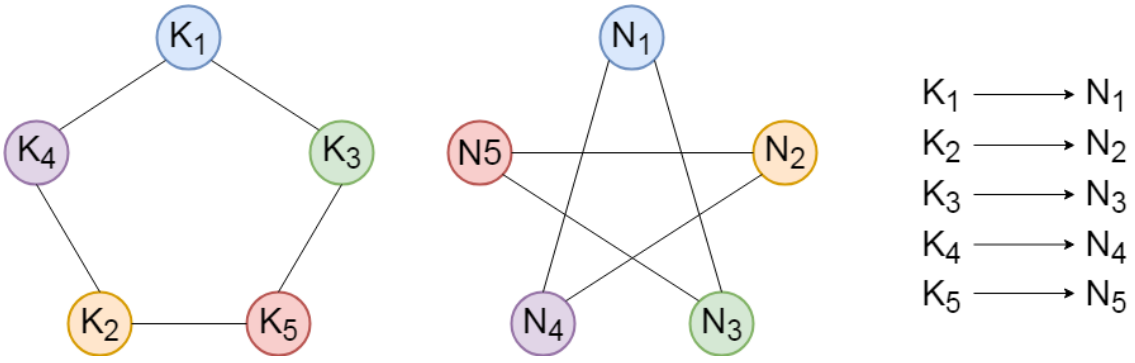
Definicija 14 Za date grafove $G = (V, E)$ i $G' = (V', E')$, *morfizam* u oznaci $f: G \to G'$, $f = (f_V, f_E)$ se sastoji od dve funckije $f_V: V \to V'$ i $f_E: E \to E'$. Morfizam f je injektivan (ili surjektivan) ako su obe funkcije f_V i f_E injektivne (ili surjektivne).

Definicija 15 Funkcija f koja graf $G = (V, E)$ preslikava na graf $G' = (V', E')$ naziva se *homomorfizmom* iz G u G', u oznaci $f: G \to G'$, ako važe sledeća svojstva:

1. Ako je $v \in V$, tada $f(v) \in V'$, odnosno $f(V) \subseteq f(V')$.
2. Ako je $e \in E$, tada $f(e) \in E'$, odnosno $f(E) \subseteq f(E')$.
3. Ako su čvorovi a i b incidentni grani e u grafu G, tada su *f(a)* i *f(b)* incidentni grani *f(e)* u grafu G'.

Definicija 16 Homomorfizam $f: G \to G'$ je *izomorfizam* ako su $f: V \to V'$ i $f: E \to E'$ bijektivna preslikavanja. Ako je $f: G \to G'$ *izomorfizam*, tada se za grafove G i G' kaže da su *izomorfni*.

Intuitivno, izomorfizam se može razmatrati kao preimenovanje čvorova i grana grafa G. Pošto ćemo koristiti grafove sa labelama, u našoj implemetaciji će od interesa biti samo grafovi čiji čvorovi imaju labele dok grane neće nositi oznake. Stoga nas interesuje preimenovanje čvorova, a za grane će dovoljan uslov biti da postoje između odgovarajućih čvorova.



Slika 2.3: Primer dva izomorfna grafa.

Problem pronalaženja izomorfnog podgrafa definišemo na sledeći način: ako su dati grafovi G i H, potrebno je pronaći podgraf grafa G (ili sve takve podgrafove) koji je izomorfan grafu H. Za neke izbore grafova G i H moguće je da postoji eksponencijalni broj pojavljivanja H u G. U opštem slučaju ovo je NP-kompletan problem [6]. Zanimljivo je spomenuti i problem provere da li su dva fiksirana grafa G i H izomorfna, koji predstavlja specijalan slučaj problema pronalaženja izomorfnog podgrafa. Za ovaj problem nije poznato da li se može rešiti u polinomskom vremenu, ali takođe nije dokazano ni da je NP kompletan. Ipak, za mnoge specijalne vrste grafova, kao što su stabla i planarni grafovi, ovaj problem se može rešiti u polinomskom vremenu.

Implementacija algoritma *VF2* [4] za pronalaženje izomorfizma u podgrafu je korišćena tokom razvoja i u daljem tekstu sledi njen kratak opis i pseudokod.

2.2.2 Algoritam VF2

Za date grafove $G_1 = (N_1, B_1)$ i $G_2 = (N_2, B_2)$, cilj izvršavanja algoritma *VF2* je da pronađe pojavljivanje grafa G_2 unutar grafa G_1. Proces pronalaženja se odvija kroz postepenu izgradnju preslikavanja M koje uparuje čvorove iz G_1 sa čvorovima iz G_2 pod određenim predefinisanim uslovima. Preslikavanje M je izraženo skupom uređenih parova (v_1, v_2), tako da $v_1 \in G_1$ i $v_2 \in G_2$. Preslikavanje $M \subset N_1 \times N_2$ je izomorfizam ako i samo ako je M bijektivna funkcija koja čuva strukturu grafa. Preslikavanje $M \subset N_1 \times N_2$ je graf-podgraf izomorfizam ako i samo ako je M izomorfizam između grafa G_2 i podgrafa od G_1.

Algoritam *VF2* postepeno gradi konačno rešenje tako što u svakoj iteraciji dodaje odgovarajuće čvorove i grane parcijalnom rešenju preslikavanja. Koristićemo stanja, gde svako stanje s može biti pridruženo parcijalnom rešenju preslikavanja, u oznaci *M(s)*. Stanje predstavlja skup uređenih parova (v_1, v_2) takvih da v_1 pripada G_1 i v_2 pripada G_2, gde uredjeni par predstavlja preslikavanje čvora iz grafa G_1 u čvor grafa G_2. *M(s)* je podskup kompletnog rešenja preslikavanja M. Sa *M(s)* identifikuju se dva podgrafa od G_1 i G_2. Podgrafovi $G_1(s)$ i $G_2(s)$ se dobijaju odabirom čvorova iz G_1 i G_2 koji se nalaze u *M(s)* i grana koje ih povezuju. Sa $M_1(s)$, $M_2(s)$, $B_1(s)$ i $B_2(s)$ označavamo skupove čvorova grafova $G_1(s)$ i $G_2(s)$ i njihovih odgovarajućih grana. Posmatrano na ovaj način, prelaz iz stanja s u stanje s' predstavlja dodavanje uparenih čvorova (v_1, v_2) parcijalnim grafovima definisanim stanjem s.

```
PROCEDURE Match(s)
  INPUT: trenutno stanje s; M(s) za inicijalno stanje s je
     prazan
  OUTPUT: preslikavanje između dva grafa

  IF M(s) pokriva sve čvorove u G2 THEN
    OUTPUT M(s)
  ELSE
    Nađi skup parova P(s) takav da su kandidati za dodavanje u
       M(s)
    FOREACH p in P(s)
      IF pravila izvodljivosti su ispunjena za dodavanje p u
         M(s)
      THEN
        Napravi stanje s' dobijeno dodavanjem p u M(s)
        CALL Match(s')
      END IF
    END FOREACH
    Povrati strukture podataka
  END IF
END PROCEDURE Match
```

Algoritam 2.1: Pseudokod algoritama VF2

Samo mali skup stanja stanja s je *konzistentan* sa traženim morfizmom. Odnosno, postoje stanja s koja u daljem izvođenju ne dovode do traženog rešenja. Primenom određenih pravila možemo suziti prostor pretrage. Algoritam uvodi skup pravila koja održavaju konzistentnost prilikom prelaska iz stanja s u stanje s'. Ova pravila se nazivaju *pravila izvodljivosti*. Koristićemo *funkciju izvodljivosti* $F(s, v_1, v_2)$ koja ima logičku vrednost *tačno* ako par (v_1, v_2) zadovoljava sve uslove izvodljivosti. Pošto

nam je potrebno da posmatramo i atribute grafova, a ne samo njihovu strukturu, tada funkcija izvodljivosti dobija oblik: $F(s, v_1, v_2) = F_{syn}(s, v_1, v_2) \wedge F_{sem}(s, v_1, v_2)$, gde F_{syn} zavisi od strukture, a F_{sem} zavisi od atributa. F_{syn} se naziva *sintaksička*, a F_{sem} *semantička* izvodljivost.

U inicijalnom stanju s_0, funkcija preslikavanja ne sadrži nijednu komponentu, $M(s_0) = \emptyset$. Za svako međustanje s, algoritam pronalazi skup uređenih parova čvorova *P(s)* koji su kandidati za dodavanje u trenutno stanje s. Skup kandidata se može suziti primenom funkcije izvodljivosti za svaki par p koji pripada *P(s)*. Ako funkcija daje logičku vrednost *tačno*, onda je naredno stanje $s^{'} = s \cup p$, gde je $p = (v_1, v_2)$. Ceo proces se rekurzivno ponavlja za $s^{'}$. Graf se na ovaj način obilazi kao po algoritmu *DFS*.

Skup parova kandidata $P(s)$ se dobija unijom dva skupa čvorova. Prvi skup sadrži čvorove koji su povezani sa $G_1(s)$ i $G_2(s)$. Označimo sa $T_1^{out}(s)$ i $T_2^{out}(s)$ skupove čvorova koji nisu u parcijalnom preslikavanju, ali su krajevi grana koje polaze iz $G_1(s)$ i $G_2(s)$. Slično, sa $T_1^{in}(s)$ i $T_2^{in}(s)$ označavamo skupove čvorova, koje nisu u parcijalnom preslikavanju, ali čiji su krajevi grana u $G_1(s)$ i $G_2(s)$. *P(s)* se sastoji od uređenih parova (v_1, v_2), gde v_1 pripada skupu $T_1^{out}(s)$ ili skupu $T_1^{in}(s)$, dok v_2 pripada skupu $T_2^{out}(s)$ ili skupu $T_2^{in}(s)$.

Pravila izvodljivosti proveravaju validnost kandidata koji potencijalno ulazi u rešenje. Proveravamo da li su čvorovi koji se dodaju rešenju na sličan način povezani u odgovarajućim grafovima G_1 i G_2 sa ostalim čvorovima u ta dva grafa. Potrebno je uporediti ulazne i izlazne stepene čvorova oba grafa. Ako za datog kandidata $(v_1, v_2) \in P(s)$ važi da je izlazni stepen čvora $v_2 \in G_2$ veći od izlaznog stepena čvora $v_1 \in G_1$ tada je kandidat (v_1, v_2) nevalidan. Slično, kandidat se validira i prema ulaznim stepenima čvorova. Kada je u pitanju semantička provera možemo uzimati u obzir labele grana i labele čvorova. Labele mogu biti numeričke ili simboličke vrednosti. U našem slučaju prilikom provere semantičke izvodljivosti mi ćemo obraćati pažnju samo na labele čvorova, dok će za grane biti dovoljno da povezuju odgovarajuće čvorove, a njihove labele nećemo uzimati u obzir.

Primer izvršavanja algoritma VF2

Ilustrujemo jednu iteraciju izvršavanja algoritma VF2 nad grafovima G i H datim na slici 2.4, tako da tražimo graf H kao podgraf grafa G.

Plavom, žutom i zelenom bojom označena su preslikavanja čvorova. U primeru polazimo od toga da je preslikavanje čvora g_1 u čvor h_1 već dodato u skup parova preslikavanja. Tokom jedne iteracije izvršavaju se naredni koraci:

1. Skup uređenih parova preslikavanja *M(s)* trenutno sadrži samo uređeni par (g_1, h_1). Algoritam nije još pronašao graf H u grafu G, nastavlja se sa pretragom.

2. Pronalazimo skup uređenih parova *P(s)* koji predstavljaju skup kandidata za dodavanje u preslikavanje M. Tražimo skupove čvorova koji su povezani sa čvorovima iz *G(s)* i *H(s)*.

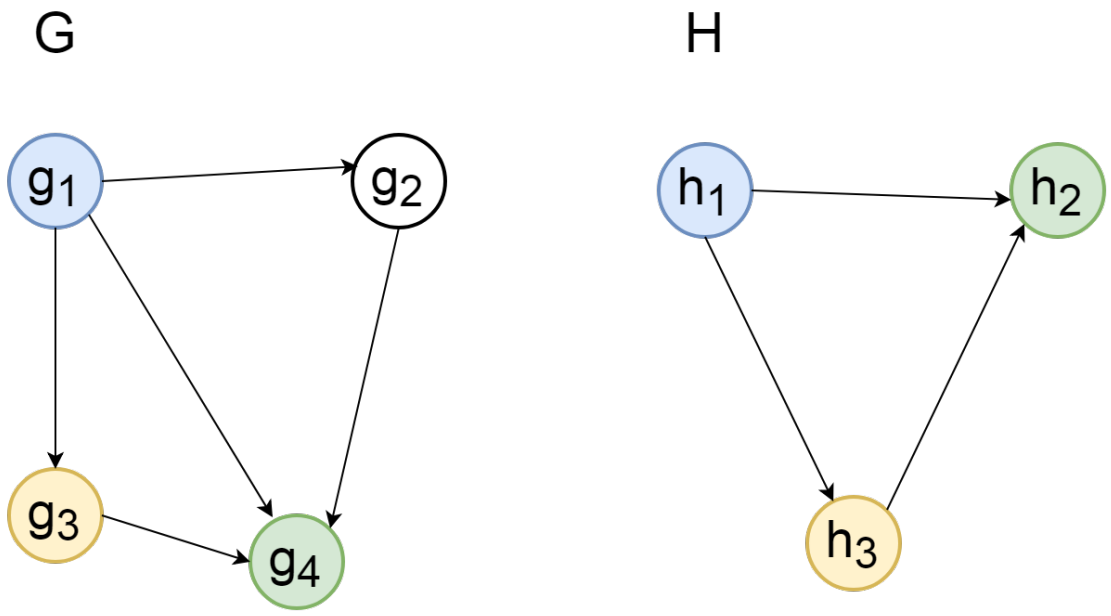
   - U skup $T_G^{out}(s)$ stavljamo čvorove koji nisu u parcijalnom rešenju, ali su krajevi grana koji polaze iz *G(s)*. Dodajemo čvorove g_2, g_3 i g_4.
   - U skup $T_H^{out}(s)$ stavljamo čvorove koji nisu u parcijalnom rešenju, ali su krajevi grana koji polaze iz *H(s)*. Dodajemo čvorove h_2 i h_3.

- U skup $T_G^{in}(s)$ stavljamo čvorove koji nisu u parcijalnom rešenju, ali čiji krajevi grana završavaju u *G(s)*. Primer nema ovakvih čvorova, skup ostaje prazan.
- U skup $T_H^{in}(s)$ stavljamo čvorove koji nisu u parcijalnom rešenju, ali čiji krajevi grana završavaju u *H(s)*. Primer nema ovakvih čvorova, skup ostaje prazan.

Konstruišemo *P(s)* dodavanjem uređenih parova (v_1, v_2), gde v_1 pripada skupu $T_G^{out}(s)$ ili $T_G^{in}(s)$, dok v_2 pripada skupu $T_H^{out}(s)$ ili $T_H^{in}(s)$. Naš skup sadrži naredne elemente: (g_2, h_2), (g_2, h_3), (g_3, h_2), (g_3, h_3), (g_4, h_2) i (g_4, h_3).

3. U ovako dobijenom skupu *P(s)* tražimo kandidata za dodavanje u preslikavanje M i na taj način prelazimo iz stanja s u stanje s'. Nad svakim kandidatom vršimo proveru pozivanjem funkcije izvodljivosti koja daje vrednost *tačno* za par koji se može dodati preslikavanju. Osnovni kriterijumi funkcije su broj ulaznih i izlaznih grana, kao i pravilan smer grana za čvorove u paru.

4. Prvi validan par je (g_2, h_2) koji se dodaje stanju s i time dobijamo stanje s'. Naredna iteracija se izvršava za parcijalno preslikavanje $M(s')$.



Slika 2.4: Grafovi G i H.

3

Metod dvostrukog potiskivanja

3.1 Uvod

Algebarski pristup grafovskim gramatikama je razvijen na Tehničkom Univerzitetu u Berlinu ranih sedamdesetih godina. Cilj je bio generalizovati gramatike Čomskog od niski na grafove. Osnovna ideja je bila generalizacija konkatenacije stringova na konstrukciju "lepljenja" za grafove. Ovo daje mogućnost da se formuliše korak prezapisivanja grafa sa dve konstrukcije lepljenja. Pristup se naziva "algebarski" jer se grafovi smatraju specijalnom vrstom algebri, a lepljenje za grafove je definisano "algebarskom konstrukcijom" koja se naziva *potiskivanje* (eng. *pushout*) u kategoriji grafova i grafovskih morfizama. Ideja dozvoljava da se primene rezultati iz algebre i teorije kategorija u algebarskoj teoriji grafovskih gramatika.

U narednom poglavlju uvodimo osnovne definicije teorije kategorija, koje će biti korišćene u daljem izlaganju.

3.2 Teorija kategorija i osnovne definicije

Kategorija je sistem međusobno povezanih objekata. Osnovna ideja je da se objekti posmatraju prema sličnosti sa drugim objektima i da na osnovu toga formiraju jednu kategoriju. Povezanost, odnosno sličnost, se opisuje preko strelica (morfizama) koje predstavljaju preslikavanje među objektima neke kategorije. Ovo će nam pomoći prilikom analiziranja sličnosti grafova i definisanja njihovih transformacija. Iz tog razloga izlažemo definicuju kategorije.

Definicija 17 Kategorija C se sastoji od:

1. Skupa Ob koji sadrži entitete koji se nazivaju *objekti*.
2. Skupa $Morf$ koji sadrži entitete koji se nazivaju *morfizmi(strelice)*.

Za skupove Ob i $Morf$ važi sledeće:

1. Za svaki morfizam (strelicu) $f \in Morf$ data su dva objekta *domen(izvor)* $s(f)$ i *kodomen(meta)* $t(f)$. Morfizam zapisujemo u oznaci $f : A \to B$, gde $A, B \in Ob$, kako bi naznačili izvor $A = s(f)$ i metu $b = t(f)$.
2. Ako su data dva morfizma $f : A \to B$ i $g : B \to C$, takva da $t(f) = s(g)$, onda je morfizam $g \circ f : A \to C$ *kompozicija* od f i g.

3. Za svaki objekat $A \in Ob$ postoji morfizam $I_A : A \to A$ ko se zove morfizam identiteta.

Dodatno, važe i naredne aksiome:

1. **[Asocijativnost]** $h \circ (g \circ f) = (h \circ g) \circ f$, za svako $f : A \to B$, $f : B \to C$ i $f : C \to D$
2. **[Identitet]** $f \circ I_A = f = I_B \circ f$, za svako $f : A \to B$.

Usmereni graf kao primer kategorije

Svaki usmereni graf možemo posmatrati kao jednu kategoriju. Uzimamo čvorove grafa kao objekte kategorije i sve grane kao morfizme koji su određeni početnim i kranjim čvorovima grane. Pored toga, kompozicija predstavlja konkatenaciju puteva, dok su identiteti "prazni" putevi. Ovo je intuitivan primer jedne kategorije, dok ćemo u nastavku izlaganja kategorije koristiti za definisanje potiska.

3.3 Transformacija grafova

Osnovna ideja svih metoda transformacija grafova je razmatranje *produkcijskog pravila* $p : L \to R$, gde se grafovi L i R nazivaju leva i desna strana.

Ako sa m označimo pojavljivanje leve strane, grafa L, u grafu G, onda $G \overset{p,m}{\Longrightarrow} H$ označava direktno izvođenje gde se p primenjuje na G što dovodi do izvedenog grafa H. Pojavljivanje grafa L u grafu G nazivamo još i *spoj* m. Intuitivno, H se dobija zamenom pojavljivanja grafa L grafom R u G.

Osnovna pitanja koja razlikuju metode transformacija grafova su:

1. Šta je graf?
2. Kako možemo naći L u G?
3. Kako definišemo zamenu L sa R u G?

3.3.1 Grafovi sa labelama

Prvo uvodimo klasu grafova koje uzimamo u obzir prilikom transformacija. To su usmereni multigrafovi sa labelama, grafovi kod kojih su čvorovi i grane označeni labelama iz dva različita skupa labela i između čvorova može postojati više paralelnih grana, čak i sa istim labelama. Osnovna činjenica je da grafovi i grafovski morfizmi formiraju kategoriju. Ovo omogućava da se većina definicija, konstrukta i rezultata formuliše koristeći teoriju kategorija.

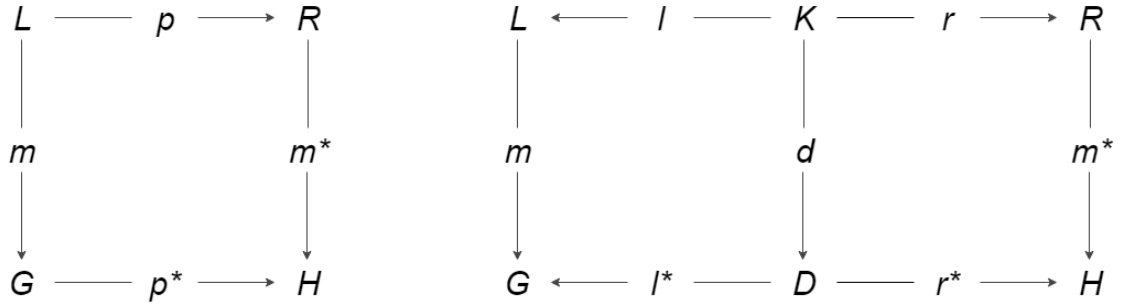
Definicija 18 Data su dva fiksirana alfabeta L_v i L_e za labele čvorova i labele grana. Graf nad L_v i L_e je uređena šestorka $G = \langle V, E, s, t, l_v, l_e \rangle$, gde je V skup čvorova, E skup grana, $s, t \colon E \to V$ su funkcije izvora i mete, $l_v \colon V \to L_v$ i $l_e \colon E \to L_e$ su labelarne funkcije za čvorove i grane.

Na slici 3.3 početni graf G_0 možemo uzeti kao primer jednog ovako definisanog grafa.

3.3.2 Spoj i produkcijska pravila

Svako produkcijsko pravilo $p : L \to R$ definiše parcijalno preslikavanje između elemenata leve i desne strane. Produkcijsko pravilo p određuje koje čvorove treba sačuvati, obrisati ili napraviti u grafu nad kojim je primenjena.

U formalnoj definiciji koja sledi, produkcijsko pravilo se definiše kao struktura $p\colon (L \xleftarrow{l} K \xrightarrow{r} R)$, gde su l i r dva morfizma, a dodatna komponenta p je ime pravila. Prilikom primene produkcijskog pravila ime nema značenje i može se zanemariti.



Slika 3.1: Produkcijsko pravilo kod metode dvostrukog potiskivanja.

Definicija 19 *Produkcijsko pravilo* $p\colon (L \xleftarrow{l} K \xrightarrow{r} R)$ se sastoji od *imena produkcije* p i para injektivnih grafovskih morfizama $l\colon K \to L$ i $r\colon K \to R$. Grafovi L, K i R se zovu: *leva strana (ls)*, *interfejs* i *desna strana (ds)* od p, redom.

Dato je produkcijsko pravilo $p\colon (L \xleftarrow{l} K \xrightarrow{r} R)$ i graf G, p se može primeniti na G ako postoji pojavljivanje L u G, grafovski morfizam koji se naziva *spoj* $m\colon L \to G$.

Ako sa slike 3.3 uzmemo produkcijsko pravilo p_1 možemo uočiti tri grafa L, K i R, koji su dati prethodnom definicijom. Graf L je leva strana produkcijskog pravila koju je potrebno pronaći u zadatom grafu, mi za te potrebe možemo uzti graf G_0 sa slike. Graf K je interfejs, odnosno graf dobijen uklanjanjem svega onoga što ne postoji u desnom grafu R, a postoji u grafu L. Na osnovu grafa K iz G_0 uklanjamo nepotrebne objekte, u našem slučaju to je samo grana $0 : e$. Na kraju, graf R nam govori šta je to potrebno dodati u graf G_0, u ovom primeru mi dodajemo grane $2 : e, 3 : E$ i čvor $2 : T$.

Definicija 20 Spoj $m : L \to G$ za produkcijsko pravilo p je grafovski homomorfizam koji mapira čvorove i grane iz L u G, na takav način da su grafovska struktura i labele očuvane.

Neformalno, spoj je preslikavanje između podgrafa datog grafa G i leve strane produkcijskog pravila koju želimo da primenimo. Kod primera sa slike 3.3, ako primenjujemo produkcijsko pravilo p_1, spoj m bi bio preslikavanje između celog grafa G_0 i grafa L produkcije p_1.

Prilikom primene produkcijskog pravila na graf G, kada se nađe spoj m moramo da obrišemo svaki objekat iz G koji se poklapa sa elementom u L (tj. u spoju m) za koji ne postoji odgovarajući objekat u R. Simetrično dodajemo u G svaki element iz R za koji ne postoji odgovarajući element u L. Svi ostali elementi iz G se čuvaju. Grubo govoreći, novi graf H je konstruisan na sledeći način: $G - (L - R) \cup (R - L)$.

Primena produkcijskog pravila može se posmatrati kao umetanje u kontekst, koji je deo datog grafa G i nije deo spoja m. Šematski opis produkcijskog pravila je dat na slici 3.1. Donji deo direktnog izvođenja je koprodukcija $p^*\colon G \to H$ koja dovodi u odnos dati i izvedeni graf vodeći računa o svim elementima koji su sačuvani direktnim izvođenjem. Simetrično m^*, koja je takođe grafovski homomorfizam, mapira desnu stranu R produkcijskg pravila na njeno pojavljivanje u grafu H.

Ako je p primenjivo na spoj m dajući graf H, onda dobijamo direktno izvođenje $G \overset{p,m}{\Longrightarrow} H$ i ono se naziva *konstrukcija dvostrukog potiska*. Primena produkcijskog pravila se može opisati izvođenjem narednih koraka:

1. Pronalazimo pojavljivanje leve strane produkcijskog pravila p u grafu G u vidu spoja $m\colon L \to G$.

2. Brišemo iz G sve objekte (slike) koji su u L, ali se ne nalaze u grafu K. Drugim rečima, tražimo graf D i morfizme d i l^* takve da oni predstavljaju potiskivanje, kao što je prikazano na slici 3.1.

3. Kao poslednji korak, umećemo desnu stranu podukcije p, graf R, u D, tačnije one objekte koji su u R, ali nisu u K.

Problematične situacije

Pre formalnog opisivanja konstrukcija dvostrukog potiska izlažemo neke problematične situacije:

1. Brisanje čvora posle kog ostaje deo grane koji "visi".

2. Produkcijsko pravilo u isto vreme definiše brisanje i čuvanje nekog čvora.

Kod metode dvostrukog potiskivanja direktna izvođenja se modeluju lepljenjem konstrukcija grafova, koja se formalno karakterišu kao potiskivanja u odgovarajućim kategorijama gde su grafovi objekti i grafovski homomorfizmi (totalni ili parcijalni) kao strelice. Produkcijsko pravilo, kod metode dvostrukog potiskivanja, je zadata parom $L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ grafovskog homomorfizma od zajedničkog grafa interfejsa K i direktno izvođenje se sastoji od dva lepljenja dijagrama grafova i totalnog grafovskog morfizma. Kontekstni graf D se dobija od grafa G tako što se uklone svi elementi iz G koji imaju inverznu sliku u L, ali ne u K. Ovakvo brisanje je opisano kao inverzna operacija lepljenja. Drugo lepljenje zasniva se na ubacivanju u H svih elemenata iz R koji nemaju inverznu sliku u K.
Da bi se izbegle spomenute problematične situacije kod metode dvostrukog potiskivanja spoj m mora da ispunjava uslov primenjivanja koji se naziva *uslov lepljenja*. Ovaj uslov se sastoji iz dva dela:

1. Da bi obezbedili da D nema grane koje "vise", *viseći uslov* obezbeđuje da ako p definiše brisanje čvora iz G onda definiše i brisanje njegovih ulaznih i izlaznih grana.

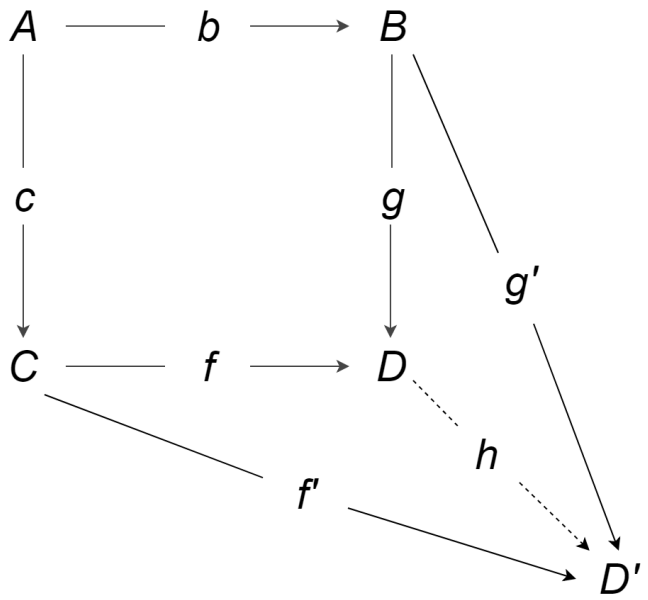
2. *Identifikacioni uslov* obezbeđuje da svaki element u G koji treba biti obrisan primenom produkcijskog pravila p ima samo jednu inverznu sliku u L. Ovo obezbeđuje da se primenom produkcijskog pravila p na G briše tačno šta je definisano tim pravilom.

Način na koji u nastavku definišemo potisak i komplement potiska će obezbediti pomenute uslove.

Definicija 21 Data je kategorija $\mathcal{C}$ i dve strelice (morfizma, preslikavanja) $b\colon A \to B$, $c\colon A \to C$ u $\mathcal{C}$, uređena trojka $\langle D, g\colon B \to D, f\colon C \to D\rangle$ sa slike 3.2 se naziva potisak od $\langle b, c\rangle$ ako:

(A) **[Komutativnost]** $g \circ b = f \circ c$

(B) **[Univerzalno svojstvo]** Za sve objekte $D^{'}$ i strelice $g^{'}\colon B \to D^{'}$ i $f^{'}\colon C \to D^{'}$ sa $g^{'} \circ b = f^{'} \circ c$ postoji jedinstvena strelica $h\colon D \to D^{'}$ takva da $h \circ g = g^{'}$ i $h \circ f = f^{'}$



Slika 3.2: Dijagram potiska i komplementa potiska.

U ovoj situaciji, D se naziva objekat potiska od $\langle b, c\rangle$. Dodatno, ako su date strelice $b\colon A \to B$ i $g\colon B \to D$, komplement potiska od $\langle b, g\rangle$ je uređena trojka $\langle C, c\colon A \to C, f\colon C \to D\rangle$ takva da je $\langle D, g, f\rangle$ potisak od $\langle b, c\rangle$. U ovom slučaju C se naziva komplement potiska od $\langle b, g\rangle$.

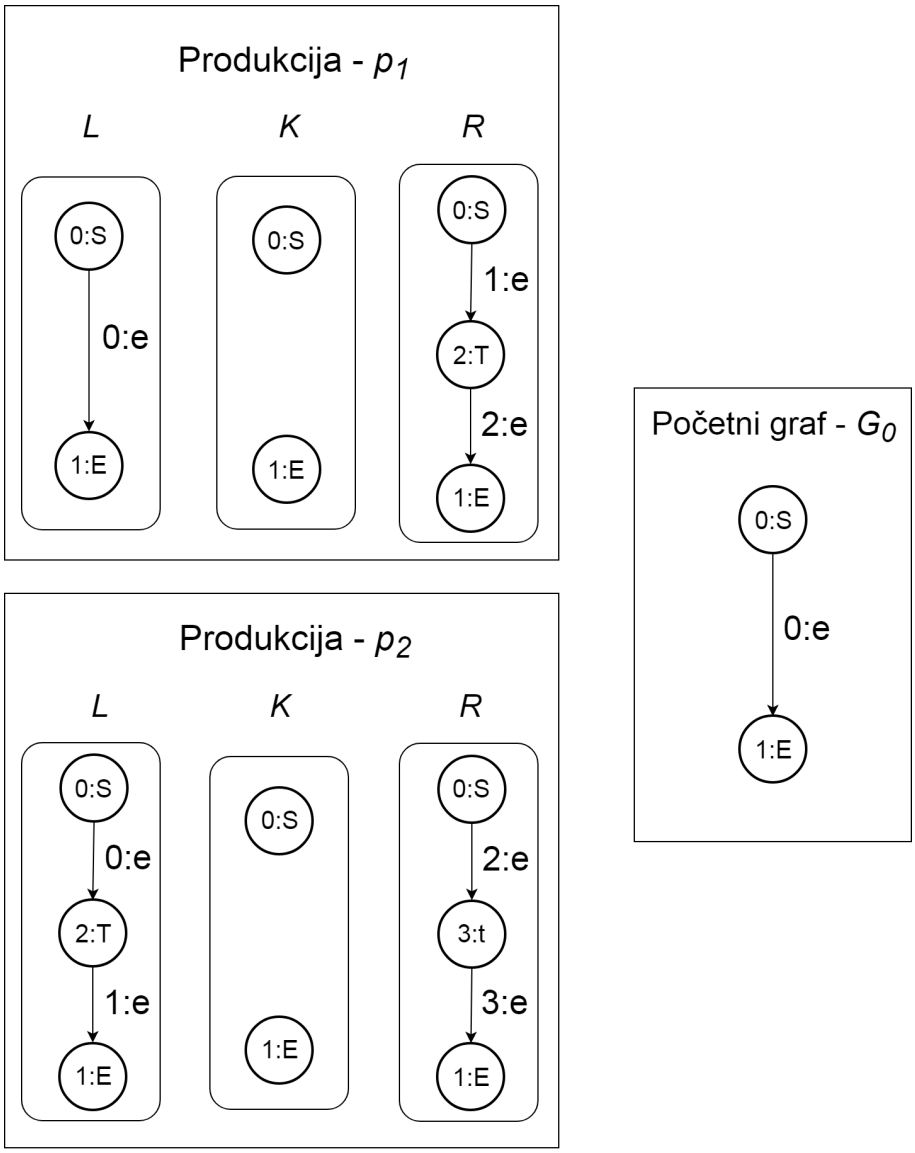
Propozicija 1 Neka su $b\colon A \to B$ i $g\colon B \to D$ dva morfizma u kategoriji grafova. Tada postoji komplement potiska $\langle C, c\colon A \to C, f\colon C \to D\rangle$ od $\langle b, g\rangle$ ako i samo ako su naredni uslovi ispunjeni:

(A) **[Viseći uslov]** Ni jedna grana $e \in D_E - g_E(B_E)$ nije susedna čvoru u $g_V(B_V - b_V(A_V))$.

(B) **[Identifikacioni uslov]** Ne postoje $x, y \in B_V \cup B_E$ takvi da $x \neq y$, $g(x) = g(y)$ i $y \notin b(A_V \cup A_E)$.

U ovom slučaju kažemo da $\langle b, g\rangle$ zadovoljava uslov lepljenja. Dodatno, ako je morfizam b injektivan, tada je komplement potiska jedinstven do na izomorfizam. Preciznije, ako su $\langle C, c, f\rangle$ i $\langle C^{'}, c^{'}, f^{'}\rangle$ dva komplementna potiska od $\langle b, g\rangle$, onda postoji izomorfizam $\phi\colon C \to C^{'}$ takav da $\phi \circ c = c^{'}$ i $f^{'} \circ \phi = f$.

Definicija 22 Dat je graf G, produkcijsko pravilo p: $(L \xleftarrow{l} K \xrightarrow{r} R)$ i *spoj* m: $L \to R$, *direktno izvođenje iz* G *u* H *korišćenjem* p *(na osnovu* m*)* postoji ako i samo ako se dijagram sa slike 3.1 može konstruisati. U ovom slučaju D se naziva *kontekstni graf* i pišemo $G \overset{p,m}{\Longrightarrow} H$.



Slika 3.3: Primer grafovske gramatike.

3.4 Grafovske gramatike

Grafovske gramatike pružaju intuitivan opis za manipulaciju grafova i grafovskih struktura koje se pojavljuju u programskim jezicima, bazama podataka, operativnim sistemima i drugim softverskim paketima. Istorijski, prvi algebarski metod grafovskim transformacijama je takozvani *metod dvostrukog potiskivanja* (eng. *Double pushout approach*). Nazvan po osnovnoj algebarskoj konstrukciji koja definiše direktan korak izvođenja, modelovano na osnovu dva mehanizma lepljenja u kategoriji grafova i totalnim grafovskim morfizmima. Navodimo definiciju grafovskih gramatika iz ugla metode dvostrukog potiskivanja.

Definicija 23 Grafovska gramatika $\mathcal{G}$ je uređeni par $\mathcal{G} = \langle (p\colon (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R))_{p\in P}, G_0\rangle$, sastoji se od familije produkcijskih pravila indeksiranih imenima u skupu P i *početnog grafa* G_0. Sekvenca izvođenja $G_0 \overset{p_1}{\Rightarrow} G_1 \overset{p_2}{\Rightarrow} ... \overset{p_n}{\Rightarrow} G_n$ sačinjava izvođenje gramatike, koje se u kraćem zapisu označava sa $G_0 \Rightarrow^* G_n$. Jezik $\mathscr{L}(G)$ generisan gramatikom G je skup svih grafova G_n tako da je $G_0 \Rightarrow^* G_n$ izvođenje gramatike.

Na slici 3.3 je dat primer grafovske gramatike koja se sastoji od dva produkcijska pravila p_1, p_2 i početnog grafa G_0. Ovaj jednostavan primer gramatike nema veliku produkcionu moć. Primenom produkcijskog pravila p_1 pa p_2 dolazi se do jedinog mogućeg rezultata, ali koristeći ovu gramatiku možemo ilustrovati bitne elemente.

4

Unity kao okruženje za razvoj

Unity je višenamenski i višeplatformski sistem za razvijanje video-igara. U poslednjoj deceniji zadobio je veliku popularnost zbog lakoće korišćenja, načina licenciranja i samog nivoa kvaliteta.

Kada je u pitanju lakoća korišćenja treba napomenuti dve stvari. Unity poseduje savremeno *WYSIWYG* razvojno okruženje koje olakšava i ubrzava rad razvijaocima igara, posebno dizajnerima koji često moraju da prave prototipove raznih koncepata. Programerima je na raspolaganju integracija sa Microsoft Visual Studio razvojnim okruženjem koje omogućava pisanje koda za video-igre ili programa koji pomažu prilikom razvoja. Za razliku od drugih sistema za razvoj, Unity podržava dva programska jezika, a to su *C#* i *Unity Script*, varijanta *Javascript* programskog jezika. Svojevremeno Unity je imao i podršku za programski jezik *Boo*, jezik inspirisan sintaksom *Python*-a, ali je podrška povučena zbog manjka zainteresovanosti.

Unity nudi nekoliko opcija kada su u pitanju licence, od kojih je jedna i besplatna. Ovakav način licenciranja daje mogućnost nezavisnim studijima da razvijaju i plasiraju svoje proizvode uz minimalne finansijske obaveze, što je jedan od većih razloga koji doprinosi njegovoj popularnosti. Ovakav trend u industriji igara prate i druge firme koje razvijaju sisteme za video-igre.

Opšti nivo kvaliteta može se opisati izlaganjem nekih od osnovnih odlika ovog sistema koje navodimo u nastavku.

4.1 Osnovne odlike

Industrija danas, od sistema za razvoj video-igara zahteva neki određeni skup odlika i funkcionalnosti koji moraju biti dostupni prilikom razvoja. Navešćemo nekoliko koji su od veće važnosti.

1. *Savremeno razvojno okruženje* - Unity poseduje razvojno okruženje koje se može koristiti na Windows i Mac platformama, sadrži veliki dijapazon alata pogodan za programere i dizajnere.

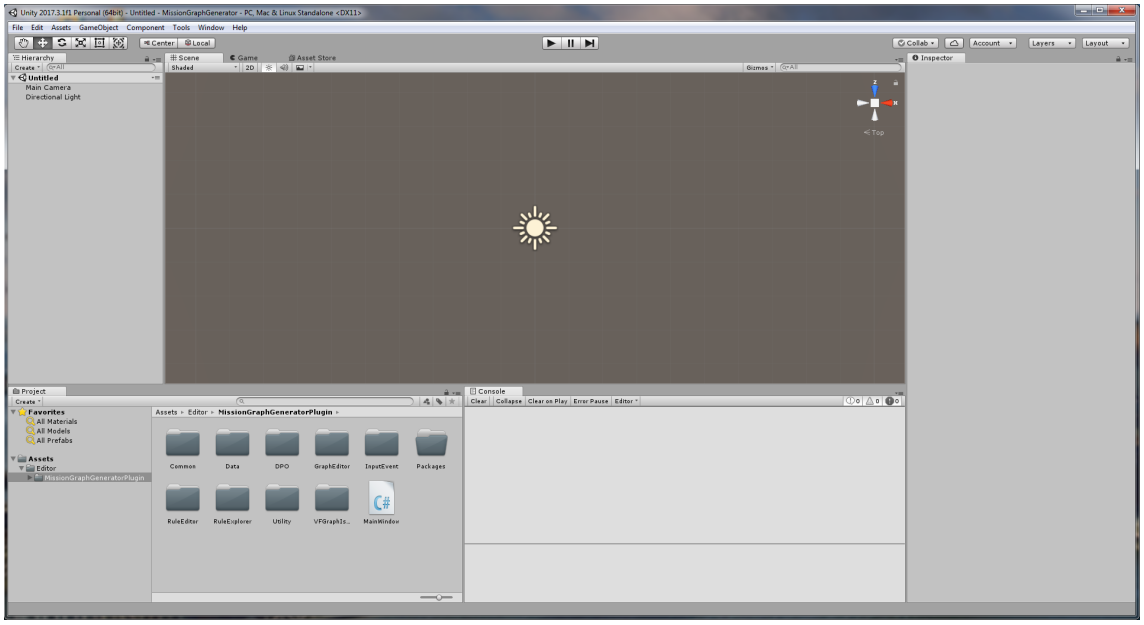
2. *Sistem za fiziku* - Visokorealistične fizičke simulacije uz podršku savremenih *NVIDIA PhysX* tehnologija.

3. *Podrška za AI navigacione sisteme* - Sistem daje mogućnost jednostavnog kreiranja navigacionih sistema za karaktere kojima ne upravlja igrač. Sistem

koristi statičku i dinamičku geometriju na osnovu kojih menja kretanje karaktera tokom igre.

4. *2D i 3D razvoj* - Omogućena je podrška za razvoj 2D i 3D igara ili drugih softverskih rešenja.

5. *Ekstenzija palete alata i razvojnog okruženja* - Jedna od važnijih osobina je mogućnost implementiranja i integrisanja novih alata unutar samog sistema. Ovo daje mogućnost ubrzavanja rada, kreiranja novih tokova rada ali i razvoj programa za rešavanje specifičnih problema.



Slika 4.1: Glavni prozor Unity sistema.

Ekstenzijom razvojnog okruženja implementiraćemo dodatak za generisanje misija. Iz tog razloga, u narednom odeljku, opisaćemo kako se to postiže u okviru sistema.

4.2 Pisanje dodataka za razvojno okruženje

Uz ispunjenje par uslova i u nekoliko koraka moguće je napraviti jednostavni dodatak za Unity sistem. Jedna od mogućnosti, koju ćemo mi koristiti, je kreiranje zasebnog prozora razvojnog okruženja. Ovaj način nam daje maksimalnu fleksibilnost prilikom dizajniranja interfejsa i razvijanja programske logike samog dodatka.

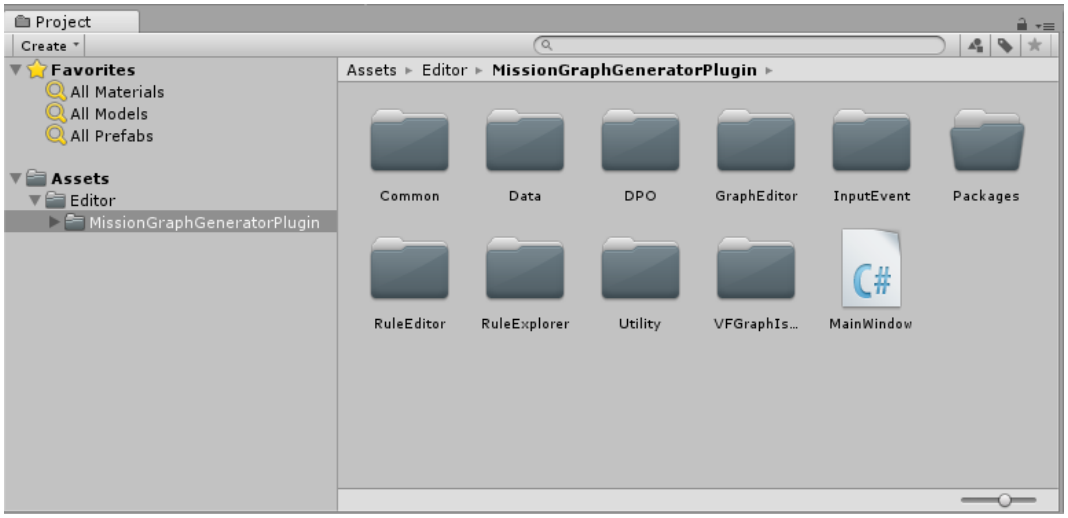
Prvi korak koji je potrebno uraditi je kreiranje *Editor* direktorijuma unutar Unity projekta. Na ovaj način razvojno okruženje zna gde da vrši pretragu naših dodataka. Neke od prednosti ovakvog pristupa:

1. Izdvajanje dodataka od ostatka strukture projekta.

2. Mogućnost nezavisnog verzionisanja i kontrole koda dodatka.

3. Jednostavna instalacija novih dodataka - dodavanjem u *Editor* direktorijum.

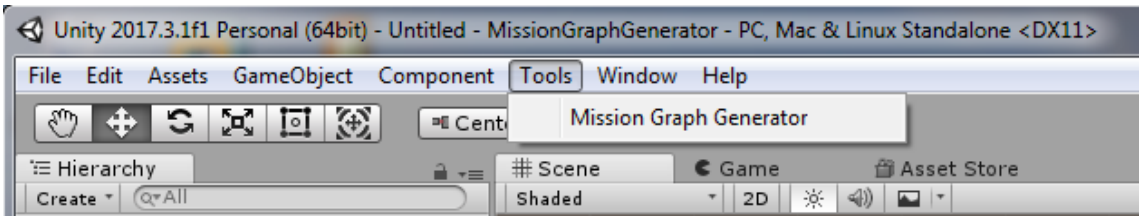
Drugi korak je kreiranje *C#* skripte koja će sadržati kod za pokretanje našeg dodatka i ona se mora nalaziti u *Editor* direktorijumu. Možemo pratiti konvenciju i fajl nazivati imenom klase koju sadrži, ali nije neophodno. Klasa može nositi bilo koji naziv, u našem slučaju to će biti *MainWindow*, pošto ova klasa predstavlja inicijalni prozor dodatka.



Slika 4.2: Primer strukture projekta.

Nakon kreiranja skripte, klasa mora da nasleđuje *EditorWindow* klasu implementiranu u Unity sistemu. Na ovaj način naša klasa nasleđuje sve potrebne osobine i metode, među kojima je i *OnGUI* metoda, koja implementira korisnički interfejs i odgovore na događaje unutar prozora.

Poslednji korak je instanciranje *MainWindow* objekta, Unity prepušta punu kontrolu za definisanje načina na koji se to radi i u kom trenutku. Prozor se može instancirati na različite akcije u okviru sistema. Tipičan primer, koji i mi koristimo, je instanciranje (otvaranje) prozora na akciju korisnika u okviru glavnog menija sistema. *@MenuItem* atributom anotiramo željenu metodu koja će biti pozvana akcijom korisnika na meni sistema. Lokacija u meniju se prosleđuje kao parametar atributa. Prilikom instanciranja prozora preporučuje se korišćenje *GetWindow* statičke metode *EditorWindow* klase. Podrazumevano ponašanje sistema je da reciklira prozore. Ponovnom akcijom korisnika na meni, postojeći prozor će biti prikazan.



Slika 4.3: Prikaz opcije za otvaranje prozora u okviru glavnog menija Unity sistema.

Renderovanje samog sadržaja prozora se postiže implementiranjem *OnGUI* metode. Unity pruža podršku za širok skup kontrola korišćenjem *EditorGUI* i *EditorGUILayout* klasa.

```
using UnityEngine;
using UnityEditor;
using System.Collections;

public class MainWindow : EditorWindow {

        [MenuItem("Tools/Mission Graph Generator")]
        public static void ShowWindow()
        {
                ApplicationData.Instance.Load();
                MainWindow mainWindow =
                EditorWindow.GetWindow<MainWindow>(
                        "MGG",
                        true
                );

                Rect mainWindowPosition = new Rect()
                {
                        x = 0,
                        y = 0
                };

                mainWindow.position = mainWindowPosition;
        }

        void OnGUI () {
                // GUI code
        }
}
```

Algoritam 4.1: Primer instancijacije prozora

5

Proceduralno generisanje grafovskim gramatikama

Proceduralno generisanje je programski pristup kreiranju sadržaja za video-igre. U zavisnosti od implementacije generatora sadržaj se može kreirati potpuno automatski, korišćenjem isključivo indirektnog korisničkog unosa. Sa druge strane imamo generatore koji generišu sadržaj uz pomoć korisnika, koji je u mogućnosti da konfiguriše ograničeni skup parametara. Ovakvi generatori obično služe dizajnerima kao pomoćni alat prilikom razvoja.

Kada pričamo o sadržaju to mogu biti različiti aspekti video-igara: mape, pravila igre, geometrija, teksture, predmeti, misije i drugi elementi. Ovde ćemo obratiti pažnju na proceduralno generiasanje misija. Prvo uvodimo neke osnovne pojmove proceduralnog generisanja, nakon toga izlažemo kako se to grafovske gramatike koriste za generisanje misija.

5.1 Osnovni pojmovi

Pojmovi *proceduralno* i *generisanje* nam naglašavaju da koristimo računarske algoritme sa ciljem da kreiramo nešto. Pod *sistemima za proceduralno generisanje* podrazumevamo sisteme koji implementiraju neki ovakav algoritam. To može biti video-igra koja sama generiše neki sadržaj ili alat koji dizajner koristi prilikom svog rada. Navešćemo neke primere sistema:

- Program koji generiše mape za potrebe igranja, bez ikakvog unosa korisnika. *Diablo* je tipičan primer igre koji sadrži ovakav sistem.
- Alat za dizajniranje mapa koji kontinualno evaluira trenutni dizajn i daje predloge za unapređenja.
- Sistem koji generiše i populiše nivoe u igricama sa vegetacijom ili drugim statičkim objektima.
- Alat za generisanje i dizajniranje vegetacije. *SpeedTree* je popularno rešenje za ovaj problem.
- Programi za proceduralno generisanje geometrije i animacije. Dobar predstavnik ovakvih sistema je *Houdini FX* [8].

5.1.1 Osobine

Kod sistema za proceduralno generisanje sadržaja postoje određene osobine koje mogu biti poželjne i one zavise od samog problema generisanja koji pokušavamo da rešimo. Jedan od slučajeva može biti generisanje vegetacije u realnom vremenu, dok drugi može biti evaluacija datog dizajna mape i davanje predloga za poboljšanje. Potrebne ili poželjne osobine se razlikuju u zavisnosti od primene. Opisaćemo neke od poželjnih osobina za takve sisteme:

- *Brzina* - Brzina generisanja zavisi od toga da li se sadržaj kreira tokom igranja ili tokom razvijanja igre. Prethodna dva primera baš ilustruju ove varijante.

- *Pouzdanost* - Neki generatori su u mogućnosti da kreiraju sadržaj koji zadovoljava širok skup kriterijuma, ali to ne mora biti važno za sve sisteme. Ako generator napravi mapu koja nema ulaz i izlaz to se može smatrati za loše generisan sadržaj. Ako ipak generator vegetacije napravi drvo koje vizuelno ne izgleda privlačno, to ne mora da poremeti samu igru ili da je napravi nekorisnom.

- *Nivo kontrole* - Određeni nivo kontrole nad generatorom je obično potreban. Tako da sam korisnik, dizajner ili igrač, može na neki način da utiče na kreiranje sadržaja. Nivo kontrole obično zavisi od praktičnih potreba i primena sistema. Kao i kod brzine, može biti važno kada se sadržaj generiše. Kontrola se obično postiže biranjem nasumičnog semena za generisanje. Drugi način je izbor skupa parametara koji kontrolišu generisanje.

- *Izražajnost i raznolikost* - Od sistema za generisanje se obično traži da mogu da generišu raznoliki sadržaj. Kako bismo izbegli da sadržaj izgleda isto ili slično, samo sa malim varijacijama. Merenje izražajnosti je veoma težak problem sam po sebi i dizajniranje generatora koji generiše raznoliki sadržaj bez negativnog uticaja na kvalitet takođe može biti komplikovano.

- *Kreativnost i verodostojnost* - Najbolji slučaj bi bio da sadržaj kreiran generatorom izgleda kao da je napravljen od strane čoveka. U većini slučajeva to ne prolazi potpuno neprimetno.

5.1.2 Klase

Proceduralne generatore možemo razvrstavati u više različitih klasa u zavisnosti od njihovih, već spomenutih, osobina i drugih odlika koje mogu posedovati.

- *Online i Offline sistemi.* Pod *online* sistemima podrazumevamo generatore koji su u sposobnosti da kreiraju sadržaj tokom igranja igre. Na ovaj način možemo dobiti igre koje nemaju kraj i menjaju sadržaj na osnovu načina igranja. *Offline* generatori se koriste tokom razvijanja video-igre ili pre početka nivoa, tokom učitavanja video-igre. Ovakvi sistemi se obično koriste za generisanje kompleksnih okruženja i mapa.

- Sistemi za generisanje neophodnog i opcionog sadržaja. Sadržaj koji je potreban za početak, razvijanje i kraj nivoa smatra se neophodnim. Sa druge strane opcioni, ili pomoćni, sadržaj se obično može odbaciti i zameniti sa

nekim drugim. Primer ovakvog sadržaja bi bila vegetacija u igri. Osnovna razlika između ova dva je što neophodni sadržaj mora uvek biti ispravan dok isto ne mora da važi za opcioni (kao kod prethodno napomenutog primera sa drvetom).

- *Generički i prilagodljivi sistemi.* U slučaju kada se ponašanje igrača ne uzima u obzir prilikom kreiranja sadržaja, tako dobijen sadržaj nazivamo *generičkim sadržajem.* Suprotno tome, kada analiziramo igranje i ponašanje igrača i to koristimo kao jedan od parametara generisanja, tada dobijamo *prilagodljivi sadržaj.*

- *Stohastički i deterministički sistemi.* Deterministički sistemi dopuštaju kreiranje istog sadržaja ako je kao ulaz dat isti skup parametara, dok kod stohastičkih sistema ponovno kreiranje istog sadržaja nije moguće.

- *Konstruktivni i generiši i testiraj sistemi.* Konstruktivni sistemi kreiraju sadržaj u jednom prolazu. Sa druge strane "generiši i testiraj" sistemi to rade tako što smenjuju faze generisanja i testiranja dobijenog rešenja. Kada dobiju rešenje koje je zadovoljavajuće, u odnosu na neke definisane kriterijume, tada se zaustavljaju.

5.1.3 Motivi za upotrebu proceduralnog generisanja

Sad kad znamo osobine proceduralnih generatora vredi pomenuti motive za njihovu upotrebu i videćemo da ih postoji više.

Do sada nam je jasno da sadržaj dobijen nekim sistemom za generisanje može biti veoma raznovrsan ili nepredvidiv, posebno kada su u pitanju stohastički sistemi. Možda, ne tako očigledno, sistemi za generisanje mogu dovesti do novih kreativih otkrića. Preciznije, algoritamski pristup kreiranju može generisati sadržaj koji je radikalno drugačiji od onog koji bi čovek napravio. Van domena igrica ovakav pristup se naziva *evolutivni dizajn.*

Kombinovanjem metoda proceduralnog generisanja i korisničkog ponašanja mogu se napraviti igre koje menjaju svoje elemente ili pravila u odnosu na to kako neko igra igru. Dobar primer za ovako nešto je igra *Left 4 Dead* [10] koja menja stanje igre, broj neprijatelja i njihovu jačinu, u odnosu na ponašanje igrača.

Možda je najbitniji motiv za generisanje sadžaja ubrzanje razvoja video-igre. Cilj može biti i uklanjanje potrebe za sadržajem napravljenim od strane čoveka. Kako ova varijanta nije najpoželjnija za ljude koji se bave pravljenjem sadržaja, možemo reći da alati koji asistiraju u generisanju i dizajniranju sadržaja mogu biti jako poželjni. Korišćenjem takvih alata razvojni timovi bi bili u mogućnosti da za kraće vreme prozvedu potrebnu količinu sadržaja za jednu video-igru.

5.2 Misije i grafovske gramatike

Definicije grafova i grafovskih gramatika smo videli u prethodnim odeljcima. Sada ćemo uvesti pojam misije u video-igrama i objasnićemo kako se to grafovske gramatike koriste za njihovo generisanje.

5.2.1 Misije u video-igrama

Video-igre su obično podeljene u nekakve celine, koje linearno slede jedna za drugom. Nalik na poglavlja u knjigama. Jedna takva celina se obično naziva *nivo*. Jedan od razloga za ovakvu podelu je da igre mogu biti toliko velike da se moraju parcijalno učitavati i izvršavati. Nivoi uglavnom imaju jasno definisan početak i kraj, kao i uslove koje treba ispuniti da bi nivo mogao da se završi. Ne tako retko, nailazimo na nivoe kojima se težina igranja povećava kako napredujemo kroz igru.

Nivo nije jedinstvena celina već se može razložiti na elemente. Preciznije, to su *misija* i *prostor*. Misija predstavlja skup zadataka koje igrač može ili mora da uradi kako bi ispunio uslove za njen uspešan završetak. Sa druge strane, prostor je okruženje sastavljeno od geometrijskih objekata, dvodimenzionalnih ili trodimenzionalnih, kroz koje igrač može da se kreće u cilju ispunjavanja uslova misije. Za misije je jako bitno da pratimo redosled zadataka, njihovu uslovnost, uzročnost i posledičnost. Kod prostora moramo voditi računa o povezanosti njegovih delova, razdaljine i razmere, kao i o postavljanju smislenih znakova koji vode igrača kroz nivo.

Kako bismo generisali smisleni nivo potrebno je obratiti pažnju na sve elemente misije i prostora, kao i uklopiti ih međusobno. Postoje različiti načini za generisanje nivoa video-igre. Jedan takav pristup je generisanje misije na osnovu koje se naknadno generiše prostor za igru. Ovaj način generisanja je pogodan za video-igre *akciono-avanturističkog* žanra ili video-igre u kojima je akcenat na naraciji, gde je bitno obezbediti jasnu povezanost u priči. Moguće je izvršiti generisanje i u suprotnom smeru, takav pristup je obično karakterističan za video-igre strateškog tipa, jer je potrebna konzistentna arhitektura i geometrija. Na ovaj način postoji rizik da geometrija i prostor koji su generisani ne pružaju veliki potencijal za generisanje poželjne misije. Pored toga, nivo kontrole nad generisanjem misije je obično niži u odnosu na druge pristupe. Iz tog razloga se ovakav pristup koristi samo za video-igre određenog tipa.

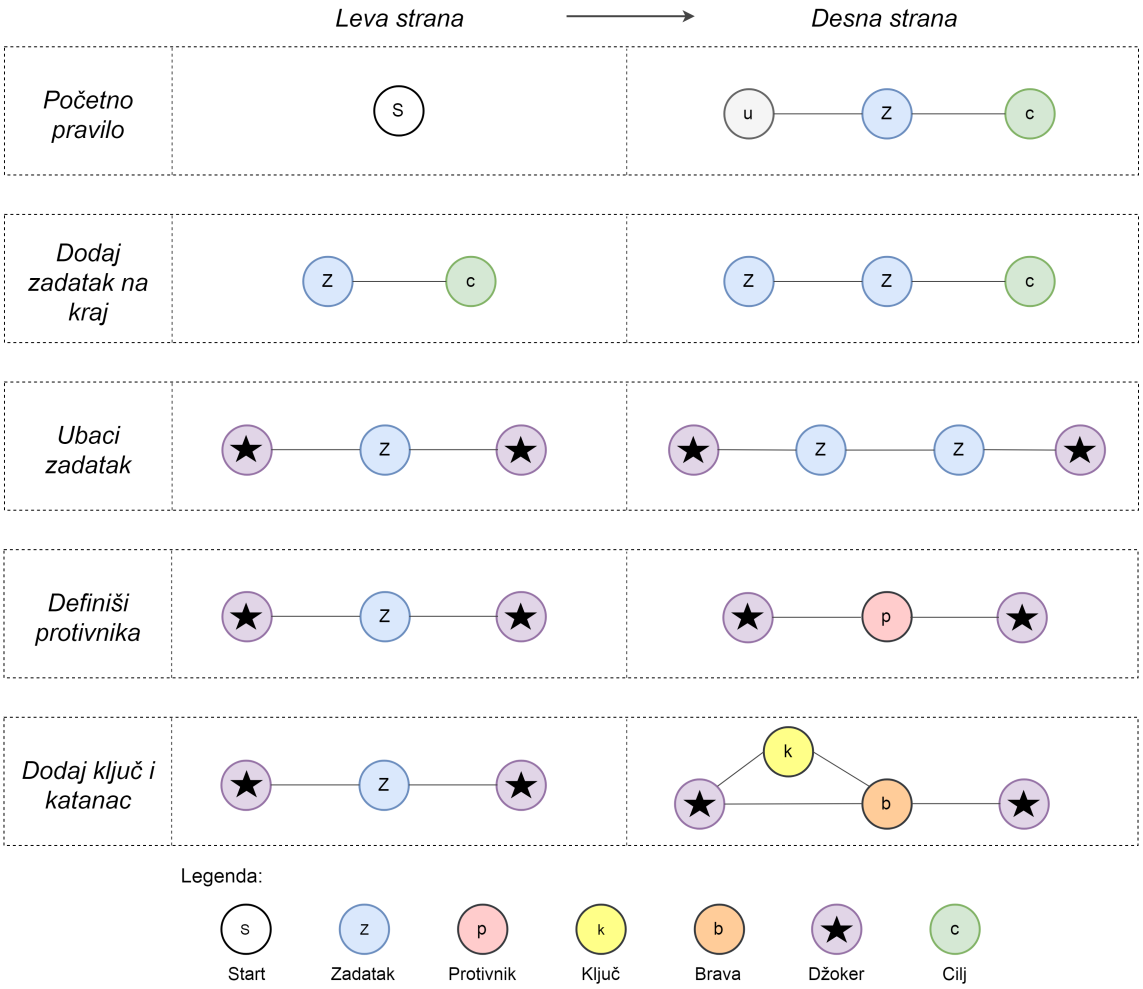
5.2.2 Generisanje misija grafovskim gramatikama

Da bismo generisali misiju korišćenjem grafovskih gramatika potrebno je da prvo definišemo alfabet gramatike. Navešćemo jednostavan primer alfabeta koji se sastoji od sledećih čvorova:

1. Početni čvor S - početni simbol od kog gramatika generiše misiju.
2. Ulaz u - početna pozicija za igrača.
3. Zadatak Z - zadatak koji je potrebno da igrač uradi.
4. Katanac b - brave koje moraju da se otključaju kako bi igrač nastavio dalje.

5. Ključ k - ključ koji otvara određeni katanac.
6. Protivnik p - protivnik kog igrač mora da pobedi.
7. Džoker ★ - džoker čvor koji menja bilo koji drugi čvor prilikom primene pravila.
8. Cilj c - dolaskom do cilja igrač završava misiju.

Kao i kod gramatika koje manipulišu stringovima i ovde imamo terminalne i neterminalne čvorove. Neterminalne čvorove obično označavamo velikim slovima, a terminalne malim. Različita pravila se mogu napraviti nad ovakvim alfabetom, u nastavku navodimo primer.



Slika 5.1: Primer grafovske gramatike i njenih pravila prezapisivanja.

Prvo što možemo primetiti jeste da može biti teško kontrolisati pravila jedne gramatike. Ovako definisanim pravilima, neka se mogu primeniti više puta, kao na primer pravilo dodavanja zadatka. Tako da broj zadataka može da se kreće od nule pa na više, zbog nedostatka definisane granice. Jedan od načina da se uvede kontrola je odabir redosleda primene pravila i broj primenjivanja. Ipak, drugačiji pristup se koristi u praksi gde proces generisanja posmatramo u delovima.

Prethodni primer je prilično jednostavan. Kako bismo dobili nešto kompleksnije moramo koristiti više pravila. Osnova strategije pri kreiranju dobrih gramatika je

da se proces razbije u više koraka. Svaki od koraka posmatramo kao jedan korak u procesu dizajniranja. Jedan korak može da generiše generalnu specifikaciju misije, dok naredni koraci mogu da se bave grupama sitnijih detalja. Praktično gledano, prvi korak bi mogao da konstruiše nasumičan graf, ali prateći određeni skup pravila. Dok u drugom koraku, skup pravila bi mogao da rekonstruiše dobijeni graf u nešto što ima smisla iz perspektive igre.

Naredni primer opisuje jedno od mogućih razlaganja, gde bi svaki korak mogao da koristi drugi skup pravila:

1. Generišemo misiju određene veličine i nasumično biramo između katanaca, ključeva i zadataka koji popunjavaju prostor između ulaza i cilja.

2. Obezbeđujemo da je redosled ključeva i katanaca ispoštovan, tačnije da se za svaki katanac prethodno može naići na ključ. Možemo i dodeliti više ključeva jednom katancu.

3. Vršimo pomeranje katanaca, ključeva i zadataka. Ovaj korak ne mora biti neophodan. Nasumično generisan graf u prvom koraku može imati manje smisla iz perspektive igre. Nekoliko katanaca ili ključeva zaredom, može dovesti do toga da igra bude dosadna ili frustrirajuća. U ovom koraku skup pravila bi mogao da obezbedi da ne dolazi do takvih slučajeva.

6

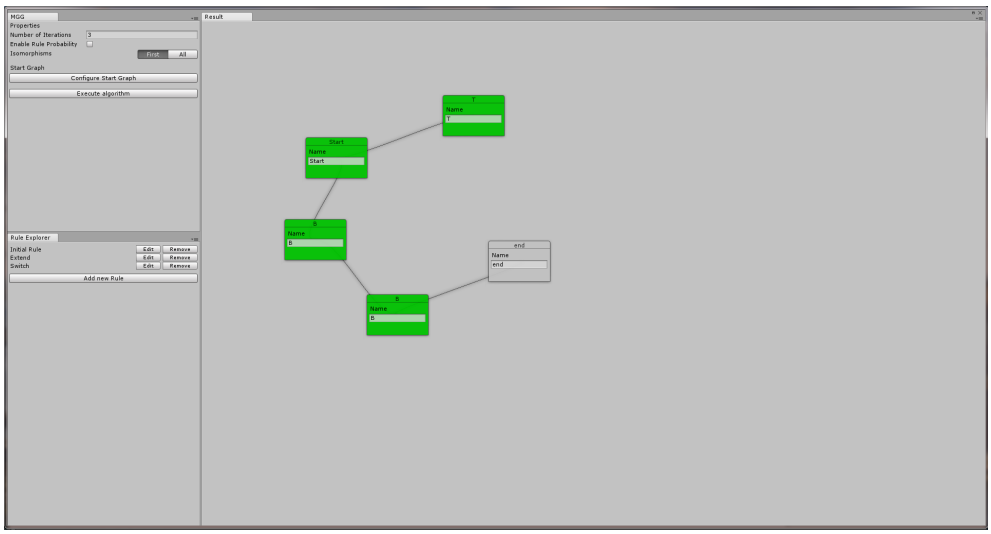
Unity dodatak za generisanje misija

Koristeći mogućnost uvođenja novih dodataka u Unity sistem, napravili smo rešenje za generisanje grafova misija za video igre. Dodatak nam daje mogućnost definisanja grafovske gramatike, početnog grafa i izvršavanje prezapisivanja grafova zasnovanog na metodu dvostrukog potiskivanja. Pored toga koristimo algoritam *VF2* za pronalaženje izomorfizma grafova. Podržavamo određeni broj parametara algoritma kako bi korisnik mogao da konfiguriše i kontroliše samo izvršavanje. Predstavićemo interfejs i korišćenje dodatka tako što ćemo opisati sve njegove prozore i funkcionalnosti. Dodatno, obratićemo pažnju na najbitnije delove implementacije.

6.1 Glavni prozor

Glavni prozor se sastoji iz sledećih komponenti:

1. Sekcije za konfigurisanje i kontrolu izvršavanja algoritma.
2. Sekcija za definisanje pravila gramatike.
3. Sekcije *Result*.



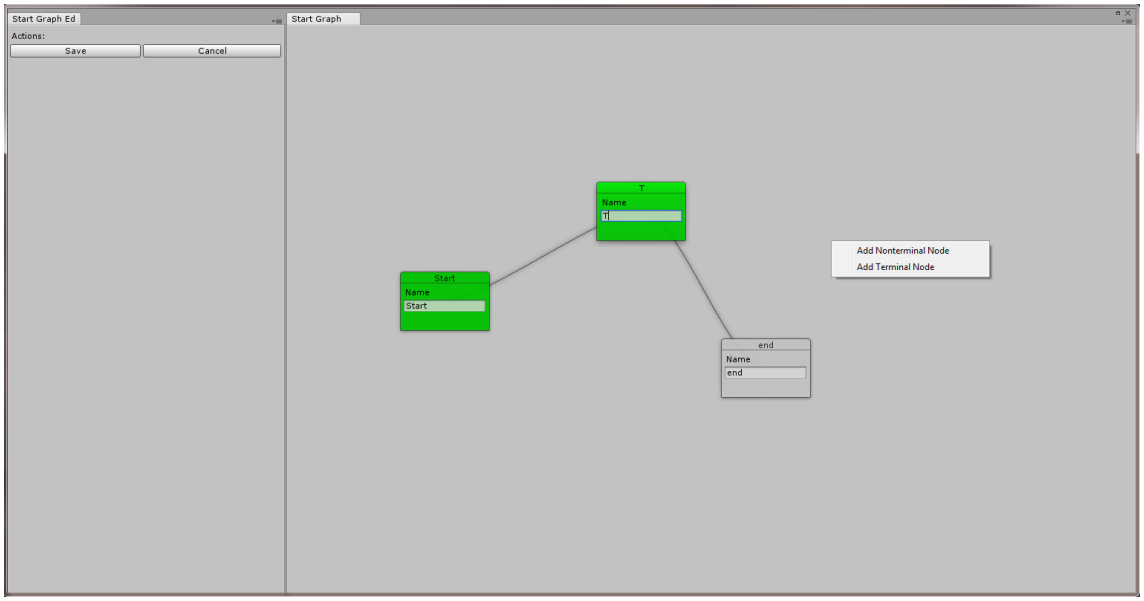
Slika 6.1: Glavni prozor dodatka.

6.1.1 Konfigurisanje algoritma

Početni graf

Osnovna opcija za konfiguraciju algoritma je početni graf. Podrazumevano, on sadrži jedan čvor, pod imenom *Start*. Korisnik je u mogućnosti da vrši bilo kakve izmene nad ovim grafom. Nije u obavezi da sadrži *Start* početni čvor, već to može biti bilo kakav proizvoljni graf. Definisanje početnog grafa se odvija u zasebnom prozoru i on pruža sledeće opcije:

- *Dodavanje čvorova* - vrši se otvaranjem kontekstnog prozora koji daje opcije dodavanja terminalnih i neterminalnih čvorova.
- *Imenovanje čvorova* - imena se mogu menjati promenom tekstualnog polja *Name* na samom čvoru.
- *Dodavanje grana* - vrši se otvaranjem kontekstnog prozora na odabranom čvoru i odabirom opcije *Make Transition*.
- *Brisanje čvorova* - omogućeno je odabirom opcije *Delete Node* iz kontekstnog prozora čvora.



Slika 6.2: Prozor za definisanje početnog grafa.

Broj iteracija

Opcijom *Number of iterations* definiše se koliki broj puta će se vršiti prezapisivanje grafa, odnosno primena pravila. Ukoliko je vrednost nula, algoritam će primenjivati pravila sve dok je to moguće. Tačnije, sve dok postoji izomorfizam sa nekom od levih strana pravila.

Prvi ili svaki izomorfizam

U slučaju odabira opcije *First Isomorphism* prezapisivanje će se izvršiti samo na prvom pronađenom izomorfizmu i ako ih možda postoji više. *All Isomorphisms* omogućava zamene svih pronađenih izomorfizama desnom stranom pravila. Ako se dva ili više pronađenih podgrafa preklapaju, prednost u zameni se daje prvom, ostali se ignorišu.

Pravila sa verovatnoćom

Ukoliko je opcija izabrana za svako pravilo se može definisati verovatnoća primenjivanja pravila, inače se pravila primenjuju redom kojim su definisana.

6.1.2 Manipulacija pravilima gramatike

Opcijama *Add*, *Remove* i *Edit* omogućena je manipulacija pravilima gramatike. Osnovni uslovi koje jedno pravilo mora da ispuni su:

1. Ime pravila mora biti jedinstveno.
2. Leva strana pravila mora da sadrži bar jedan čvor.
3. Terminalni čvorovi leve strane se moraju pojaviti i na desnoj strani pravila.
4. Joker čvorovi leve strane se moraju pojaviti i na desnoj strani pravila.

6.1.3 Pokretanje algoritma i rezultat

Nakon definisanja svih potrebnih parametara i pravila, algoritam se pokreće stiskom na dugme *Start Execution*. *Result* sekcija glavnog prozora prikazuje dobijeni graf primenom pravila na početni graf. Karakteristično za ovu sekciju je da nije omugućena manipulacija dobijenog grafa, već služi isključivo za prikaz kranjeg rezultata.

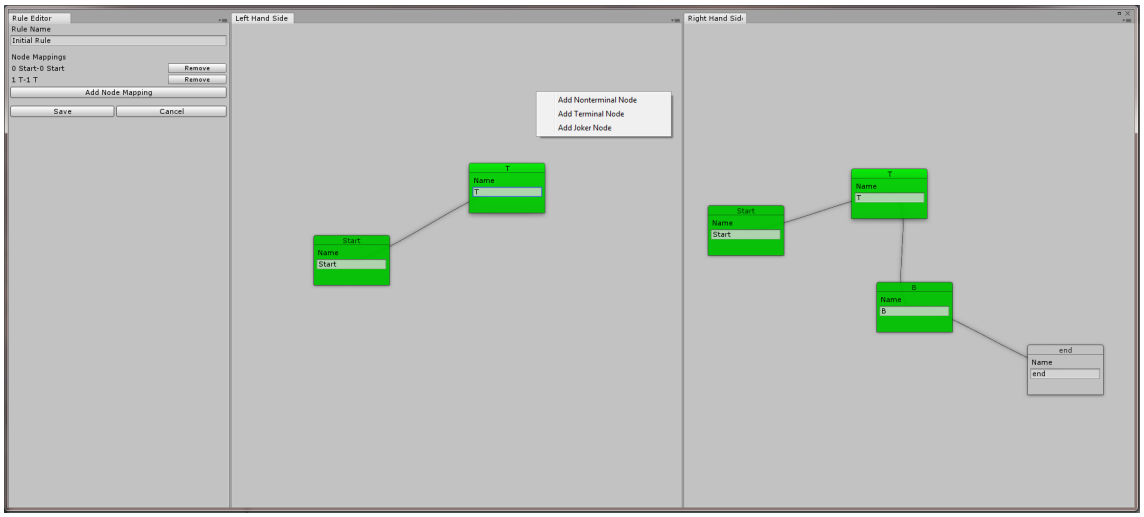
6.2 Prozor za definisanje pravila gramatike

Prozor se sastoji iz narednih komponenti:

1. Sekcija za definisanje osnovnih karakteristika pravila.
2. Dve sekcije za definisanje leve i desne strane pravila gramatike.

Definisanje leve i desne strane pravila

Leva i desna strana pravila se definišu na isti način kao i početni graf. Sve opcije su omogućene. Dodatno je potrebno mapirati čvorove leve strane na čvorove desne strane koji se ne menjaju prilikom primene pravila. Obrisani i dodati čvorovi na desnoj strani se ne mapiraju.



Slika 6.3: Prozor za definisanje pravila gramatike.

Verovatnoća

Ako je omogućen izbor pravila na osnovu njegove verovatnoće, moguće je definisati verovatnoću primene pravila. Vrednost se kreće između nula i jedan. Svako pravilo mora imati definisanu verovatnoću i ukupan zbir verovatnoća svih pravila mora biti jedan.

6.3 Renderovanje i manipulacija grafova

Do sad smo opisali nekolicinu prozora koji imaju mogućnost prikazivanja grafa. Za renderovanje i manipulaciju nad grafovima implementirana je komponenta *GraphEditorWindow*. U pitanju je klasa koja nasleđuje *Unity* klasu *EditorWindow*.

Instanca klase *GraphEditorWindow* dozvoljava manipulaciju, kao podrazumevano ponašanje i pored toga renderuje prosleđeni graf. U slučaju da graf nije definisan, pravi se instanca praznog grafa.

Klasa *Graph* se koristi kao struktura koja opisuje graf. Ista klasa se koristi kroz ceo dodatak. Nema razlike u instancama grafova koje korisnik unosi i onih koje koristi algoritam dvostrukog potiska i algoritam *VF2* .

Korisnički događaji u okviru prozora su registrovani u rečniku po njihovom tipu. Odgovarajuća metoda klase se poziva za svaki događaj u okviru *OnGUI* metode koju poziva sam sistem. Metode događaja manipulišu strukturom grafa pozivajući javne metode instance. Samim tim prozor nije svestan interne logike manipulacije nad strukturom.

Renderovanje čvorova se vrši na osnovu podataka o čvoru koji su implementirani u klasi *BaseNode*. Unity daje mogućnost korišćenja *GUI.Window* metode za kreiranje podprozora, kao jednu vrstu kontrole na interfejsu. Svaki čvor je predstavljen jednom ovakvom kontrolom. U odnosu na tip čvora, terminalni ili neterminalni, prikaz kontrole se menja kako bi korisnik mogao vizuelno da ih razlikuje.

6.4 Aplikacijski podaci

Kada je reč o modelu podataka u implementaciji dodatka, potrebno je opisati kretanje i skladištenje podataka. Klasa *ApplicationData* je implementirana po *Unikat (Singleton)* projektnom uzorku. Prilikom pokretanja dodatka pravi se jedna instanca ove klase koja je validna tokom trajanja izvršavanja. Instanca sadrži naredne podatke:

1. Početni graf.
2. Rečnik pravila gramatike.
3. Rezultat prezapisivanja.

Kako bi se svi delovi korisničkog interfejsa na vreme i adekvatno ažurirali, ova klasa implementira i *Posmatrač (Observer)* projektni uzorak korićenjem *C#* događaja. Prilikom bilo koje promene u podacima sve klase koje registruju svoju metodu na događaj će biti obaveštene o prirodi promene u datom trenutku. Na ovaj način ne postoji direktna spregnutost između raznorodnih komponenti, već svaka komponenta osluškuje promene i menja podatke koji su važni za nju.

6.5 Dvostruki potisak

Metod dvostrukog potiskivanja je implementiran kroz klasu *DPOController*. Svaka instanca predstavlja jedno izvršavanje algoritma sa prosleđenim parametrima. Prilikom instanciranja potrebno je definisati početni graf kao i pravila gramatike koja će se koristiti prilikom prezapisivanja grafa. Izvršavanje algoritma se započinje pozivanjem metode *Execute*. Nakon izvršavanja, do rezultata se dolazi pristupanjem javnom članu *ResultGraph*, koji se i koristi prilikom iscrtavanja u *Result* sekciji glavnog prozora. Predstavićemo pseudokod izvršavanja dvostrukog potiska.

```
PROCEDURE ExecuteDPO(StartGraph, RuleMap)

    INPUT: Početni graf - StartGraph, Pravila grafovske
       gramatike - RuleMap

    OUTPUT: Prezapisani graf

    RuleArray = OrderRuleMap(RuleMap);

    CurrentSolution = StartGraph;

    FOREACH rule in RuleArray

        isomorphism = FindIsomorphism(CurrentSolution,
           rule.LeftSide);

        IF isomorphism

            RemoveLHSNodes(CurrentSolution);
```

```
            RemoveObsoleteEdgesForExistingNodes(CurrentSolution);
            AddNewEdgesForExistingNodes(CurrentSolution);

            AddNewNodesFromRHS(CurrentSolution);
            InsertEdgesForNewNodes(CurrentSolution);

        END IF

    OUTPUT CurrentSolution;

END PROCEDURE ExecuteDPO
```

Algoritam 6.1: Pseudokod DPO algoritma

Redom opisujemo korake opisane pseudokodom:

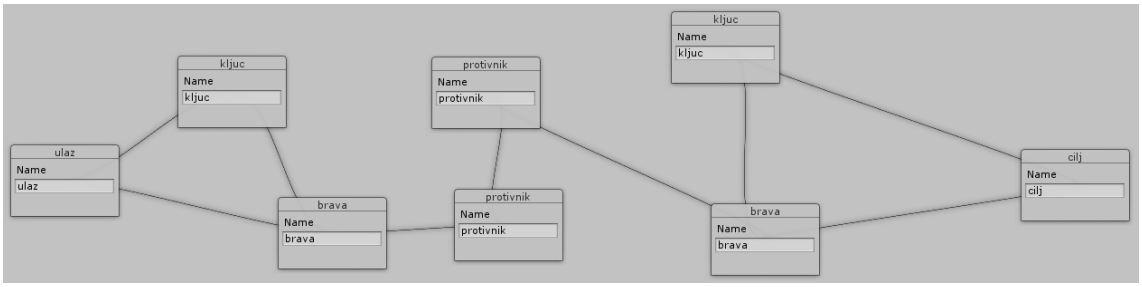
1. *OrderRuleMap* - Postavlja redosled primene pravila u zavisnosti od napomenutih konfiguracija izvršavanja algoritma.
2. *CurrentSolution* - Predstavlja tekuće rešenje izvršavanja, na početku je to početni graf.
3. *FindIsomorphism* - Korišćenjem algoritma *VF2* pronalazi izomorfizam u grafu koji se koristi za prezapisivanje trenutnog rešenja, odnosno grafa.
4. *RemoveLHSNodes* - Uklanja čvorove koji ne postoje na desnoj strani pravila prezapisivanja.
5. *RemoveObsoleteEdgesForExistingNodes* - Uklanja grane koje ne postoje na desnoj strani pravila prezapisivanja.
6. *AddNewEdgesForExistingNodes* - Dodaje nove grane, za postojeće čvrove, koje se pojavljuju na desnoj strani, ali ne i na levoj strani pravila prezapisivanja.
7. *AddNewNodesFromRHS* - Dodaje nove čvorove koji se pojavljuju samo na desnoj strani pravila prezapisivanja, slično kao i za grane u prethodnom koraku.
8. *InsertEdgesForNewNodes* - Dodaje grane za čvorove dodate u prethodnom koraku.

Navedeni pseudokod predstavlja uopšteni opis izvršavanja algoritma, sama implementacija na nekim mestima odstupa, ali potrebni koraci i njihov redosled ostaju isti.

6.5.1 Primeri misija dobijenih generisanjem

Navodimo dva primera dobijena korišćenjem dodatka u okviru Unity sistema. Prilikom generisanja korišćene su različite konfiguracije koje pruža algoritam. Koristimo grafovsku gramatiku prikazanu na slici 5.1.

Primer 1.



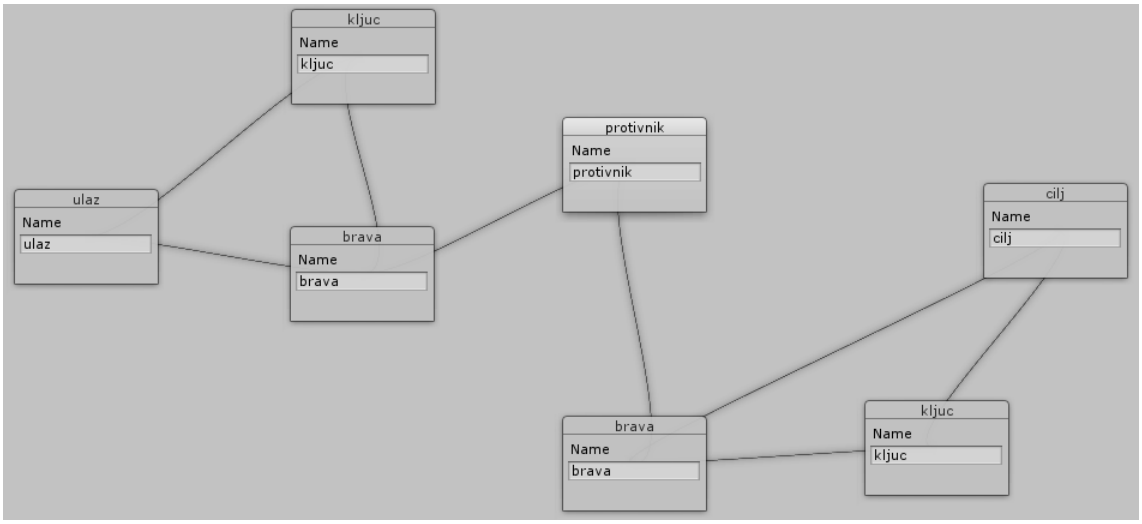
Slika 6.4: Primer misije dobijene sekvencijalnom primenom pravila.

U primeru na slici 6.4 definisani su sledeći parametri algoritma:

1. Sekvencijalno izvršavanje.
2. Pravila se primenjuju samo na prvi pronađeni izomorfizam.
3. Izvršeno je osam iteracija primene pravila.

Na osnovu dobijene misije može se uočiti određena pravilnost u dobijenom grafu. Ulaz u misiju je ograničen *ključem* i *bravom* kao i cilj misije, dok se između nalaze dva čvora koji predstavljaju *protivnike*. Rezultat je posledica sekvencijalne primene pravila zadatih gramatikom. Redosled primene pravila je isti kao redosled definisanja pravila na slici 5.1. Povećanjem broja iteracija se mogu generisati slične misije sa većim brojem čvorova. Ako bi se data gramatika koristila sa ovakvim odabirom parametra, mogli bismo jedino da generišemo misije koje dosta liče jedna na drugu, što bi proizvelo osećaj ponavljanja u video-igri. Jedina varijacija bi bila veličina misije. Ipak, gramatika i njeni parametri se mogu koristiti za generisanje samo jedne misije ili manjeg skupa misija. Ovaj pristup može izgledati čudno na prvi pogled, ali postoje slučajevi kada želimo da nekoliko uzastopnih misija imaju slične strukture, a da variraju u težini i dužini igranja. Na ovaj način igrač u prvoj misiji uči osnovne koncepte i pravila igre, dok svaka naredna misija predstavlja sve veći izazov.

Primer 2.



Slika 6.5: Primer misije dobijene sekvencijalnom primenom pravila.

U primeru na slici 6.5 definisani su sledeći parametri algoritma:

1. Probabilističko izvršavanje.
2. Pravila se primenjuju samo na prvi pronađeni izomorfizam.
3. Izvršeno je sedam iteracija primene pravila.

U ovom primeru koristimo relativno sličan skup pravila, glavna izmena je primenjivanje probabilističkog odabira pravila gramatike. Za pravila su definisane verovatnoće primene, pravilo *definiši protivnika* ima najmanju verovatnoću primene, stoga se čvor *protivnik* pojavljuje samo jednom. Kada žeimo da umanjimo broj pojavljivanja određenog pravila, probabilističko generisanje može biti od koristi. Često se određeni elementi pojavljuju ređe tokom misije, to mogu biti izazovniji protovnici ili objekti za koje je bitno da budu retki.

Data gramatika je relativno jednostavna. Možemo uočiti da su misije generisane njenom primenom prilično linearne. Ako bismo to želeli da izbegnemo, bilo bi potrebno uvesti pravila koja predstavljaju grananja u misiji. To bi nam omogućilo i eventualno generisanje misija sa više ciljeva. Drugi problem je bliskost ključeva i katanaca. Gramatiku možemo proširiti pravilima izmeštanja ključeva. Time bismo svakako povećali težinu igre i smanjili njenu predvidljivost.

7

Dalji razvoj

Kao što smo već spominjali, konstrukcija nivoa u video-igri se vrši generisanjem misije i prostora za datu misiju. Ukoliko se odlučimo da na osnovu definicije misije generišemo prostor, to možemo uraditi na više načina. Ovde ćemo opisati jednu mogućnost, koja ne samo da može da pomogne u generisanju definicije prostora već i same geometrije.

Jedan od popularnih programa za razvoj 3D animacije i geometrije, pod nazivom *Houdini FX*, zasnovan je na proceduralnom generisanju. Iz ovog razloga veoma je pogodan za razvoj raznih vizuelnih efekata, posebno u filmskoj industriji. Poslednjih godina njegova popularnost raste i u industriji video igara, izlaskom zasebnog programa *Houdini Engine* [9]. Zapravo, on služi kao dodatak drugim programima za razvoj animacije i geometrije, ali isto tako i sistemima za razvoj igara. Na ovaj način nam je omogućeno da uvedemo proceduralne odlike i funkcionalnosti u Unity sistem. Kako bismo bolje dočarali njegovu moć, prvo ćemo objasniti kako *Houdini FX* definiše geometrijske objekte. To će nam biti dovoljno da shvatimo suštinu, jer i sa drugim vrstama objekata radi na veoma sličan način.

Proceduralna priroda Houdini sistema zasnovana je na operatorima. Svaki objekat, pa i geometrijski, konstruiše se konektovanjem niza operatora. Svaki operator možemo posmatrati kao jednu funkciju u programskom jeziku koja može imati veliki broj parametara. Ovo nam omogućava nelinearni razvoj, pa čak i uvođenje novih operatora koji su zasnovani na već postojećim. Preciznije, operatori formiraju mrežu kroz koju protiču podaci i svaki od operatora manipuliše tim podacima. Svaka mreža može imati jedan ili više izlaza. Izlazi mogu da predstavljaju: 3D geometriju, razne vrste slika ili tekstura, simulacije čestica, algoritme renderovanja, animaciju, zvuk, ili čak kombinaciju navedenih elemenata.

Mrežu operatora moguće je apstrahovati u digitalne pakete koji mogu da definišu parametre koji se prosleđuju mreži kao ulaz. Najvažniji motiv za ovako nešto je to što se paketi mogu koristiti unutar Houdini sistema, a posredstvom njega i u sistemima za video-igre.

To nas dovodi do sledeće mogućnosti. Graf generisan našim generatorom misija moguće je rekonstruisati u graf prostora i predstaviti ga u određenom formatu koji se može proslediti Houdini sistemu. Na osnovu informacija o prostoru, mreža operatora bi mogla da generiše kompletnu geometriju za datu misiju.

Ovakva spona između generatora misija i Houdini sistema bi mogla da bude povod za dalji razvoj. Time bi kompletno automatizovali generisanje jednog ili više nivoa za video-igru.

8

Zaključak

Sa ciljem da predstavimo pojam proceduralnog generisanja za misije, opisali smo jednu od tehnika i izložili njenu implementaciju. Tom prilikom, koristili smo grafovske gramatike i metode prezapisivanja grafova. Metodom dvostrukog potiskivanja dali smo okvir za konstruisanje gramatike i njenih pravila. Kod primene pravila gramatike, upotrebljen je algoritam *VF2* za pronalazak izomorfizma u podgrafu grafa misije. U implementaciji je korišćen *Unity* sistem i njegova mogućnost pisanja ekstenzija. Na ovaj način korisniku smo pružili intuitivan način za definisanje procesa automatskog generisanja misija u video-igri tokom njenog razvoja.

Generator misija ima za svrhu da u kretivnom smislu unapredi i ubrza rad na dizajniranju misija tokom razvoja vdeo-igre. Prednost ovog alata je što je implementiran kao dodatak *Unity* sistemu. Eventualni ulazi i izlazi dodatka, pre i posle generisanja, mogu se kretati i koristiti od strane drugih komponenti sistema, što pozitivno može uticati na sam razvoj iz ugla efikasnosti. Pored toga, metod dvostrukog potiskivanja, sama po sebi, daje intuitivniji način za opis pravila gramatike. Time dobijamo na lakoći učenja i korišćenja alata.

Trenutna implementacija daje mogućnost samo kreiranja misije u okviru nivoa video-igre. Prostor i geometrija su nešto što se naknadno mora kreirati, manuelnim ili automatskim putem, na osnovu dobijenog grafa misije. Dodatno, alat trenutno ne dopušta definisanje grupa pravila prezapisivanja, kod kojih se redosled i način izvršavanja može posebno definisati. Ova poboljšanja, kao i ona navedena u prethodom poglavlju, se mogu uzeti u obzir prilikom daljeg unapređivanja i razvoja.

Bibliografija

[1] Rozenberg, G. (1997). Handbook of graph grammars and computing by graph transformation. Singapore: World Scientific.

[2] Anderson, J. (2005). Diskretna matematika sa kombinatorikom. Beograd: CET.

[3] Shaker, N., Togelius, J. and Nelson, M. (2016). Procedural Content Generation in Games. Cham: Springer International Publishing.

[4] Cordella, L., Foggia, P., Sansone, C. and Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(10), pp.1367-1372.

[5] Tadres, A. (n.d.). Extending Unity with Editor Scripting. Packt Publishing - ebooks Account (September 2015).

[6] Cook, S. (1971). The complexity of theorem-proving procedures. Proceedings of the third annual ACM symposium on Theory of computing - STOC '71.

[7] Interactive Data Visualization, Inc. SpeedTree - Unity [softverski paket]. Dostupno na adresi: https://store.speedtree.com/unity/.

[8] Side Effects Software, Inc. Houdini FX [softverski paket]. Dostupno na adresi: https://www.sidefx.com/products/houdini-fx/.

[9] Side Effects Software, Inc. Houdini Engine [softverski paket]. Dostupno na adresi: https://www.sidefx.com/products/houdini-engine/.

[10] Valve South, Left 4 Dead [video-igra], Valve Corporation, Bellevue, WA.