Proceduralno generisanje naselja

Sadržaj

1 POZADINA

Već duže vreme u industriji igara postoji potreba za efikasnim algoritmima za proceduralno generisanje sadržaja. S naglim povećanjem hardverske moći i zahtevima igrača količina sadržaja koji treba pružiti raste i samim tim proces ručnog kreiranja postaje dugotrajniji i skuplji. Jedno od rešenja problema leži u proceduralnim algoritmima. Danas postoji jako veliki broj takvih algoritama sa najrazličitijim primenama. Nama su od značaja isključivo oni koji su upotrebljivi u igrama, tako da pre nego što počnemo moramo definisati šta želimo da dobijemo kao rezultat i na koji način. Da li nam je od interesa da sadržaj bude fizički realan? Da li rezultati moraju biti „korektni“? Odgovor je ne. Igraču će vrlo malo verovatno biti od interesa takvi detalji. Ono na šta treba usmeriti pažnju jeste koliko je taj sadržaj interesantan za igrača i kako će on reagovati kada ga vidi. Stoga će nama od najvećeg interesa biti efikasnost, varijabilnost, fleksibilnost algoritma i, naravno, estetski kvalitet generisanog sadržaja. Želimo da promenama ulaza dobijemo što veći broj različitih rezultata.

Ovde će vam biti prezentovan algoritam za proceduralno generisanje naselja. Cilj je da se dizajnira algoritam koji će da ispuni sve gorenavedene uslove. Sela moraju biti dovoljno različita, ali i u neku ruku slična. Idealno bi bilo ako bi mešanje kultura automatski proizvodilo nove arhitekture koje bi sadržale po nešto iz svake. „Realnost naselja“ u smislu lokacija građevina, puteva i sl. su takođe potrebne.

2 UVOD

Algoritam je podeljen na nekoliko faza u kojima generiše različit tip sadržaja. Na najvišem nivou je podeljen na dve faze:

1) Generisanje karakteristika naselja: populacija, društveno uređenje, razvijenost itd.
2) Geometrija naselja

U prvoj fazi ključni koncept na kome se zasniva algoritam je „karakteristika naselja“. Karakteristika predstavlja apstrakciju nečega što čini naselje. Karakteristike mogu biti populacija, vojska, industrija itd. Na osnovu nezavisnih parametara (ulaza u proceduru) se generiše niz međusobno zavisnih karakteristika. Karakteristike koje će ovde biti iznesene, kao i njihov ukupan broj, ne moraju striktno biti takvog oblika, ovo je samo jedna varijanta. Mogu se i proširiti i suziti, u zavisnosti od potreba konkretne igre. Prva faza se dalje deli na veći broj manjih koraka.

Ulaz u proceduru (nezavisni parametri) su:

1) Geografija terena: reljef, resursi, obradivo zemljište, vegetacija i voda/kopno
2) Kultura: dogmatizam/skepticizam, duhovnost/materijalnost i arhitektura
3) Istorija: ratovi, prirodne katastrofe i prethodno naselje koje je bilo na toj lokaciji

Karakteristike naselja (zavisni parametri), koji će detaljnije biti objašnjeni u nastavku su:

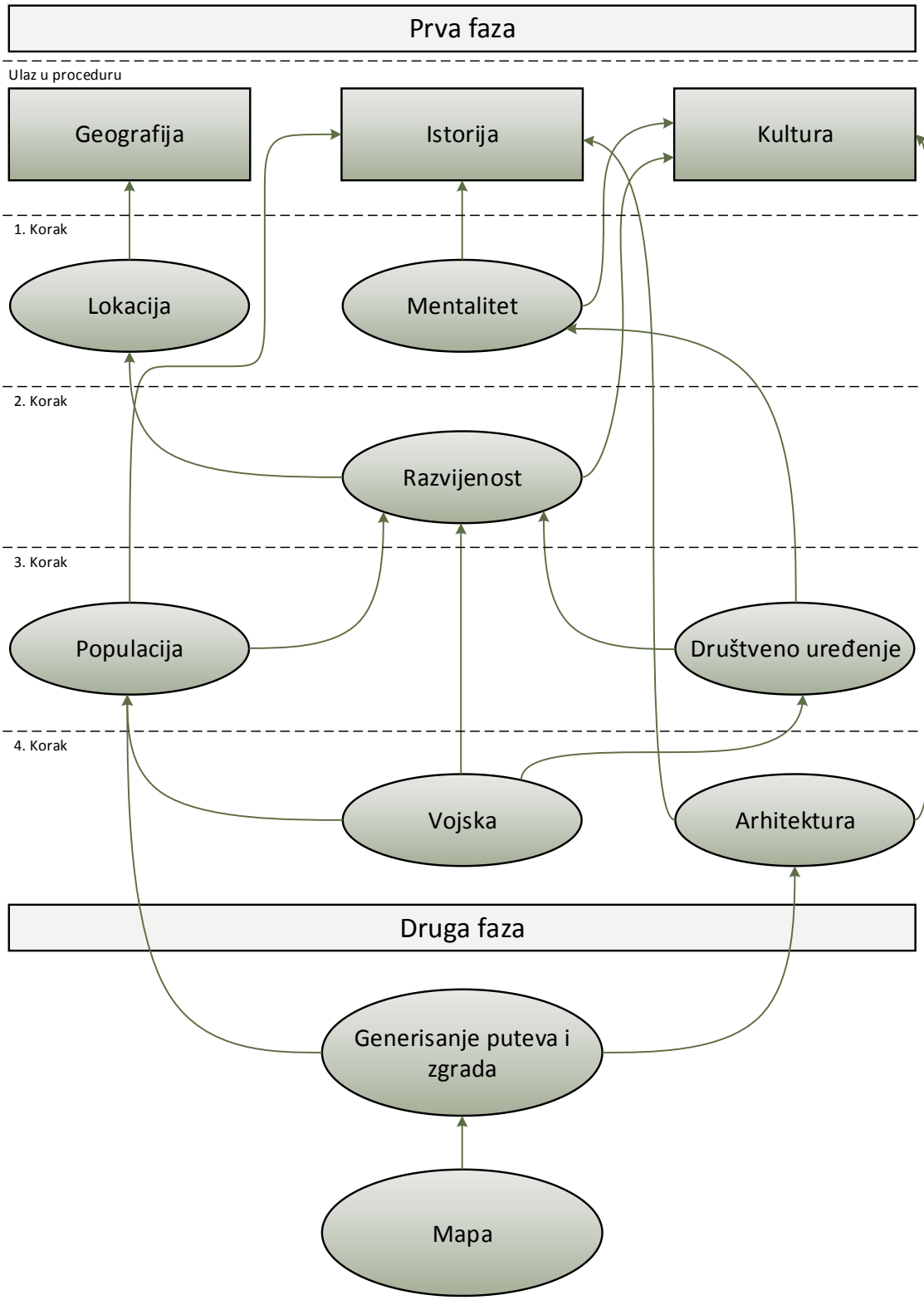
1) Lokacija
2) Arhitektura
3) Mentalitet stanovnika
4) Razvijenost
5) Društveno uređenje
6) Populacija
7) Vojska

U drugoj fazi se pozivaju algoritmi za generisanje same geometrije. Druga faza se deli na:

1) Generisanje puteva
2) Generisanje građevina

Nakon druge faze kreiranje mape naselja je trivijalan posao.

Graf zavisnosti navedenih parametara i koraci u kojima se vrši njihovo sračunavanje i generisanje je dat na slici 2.1.


Prva faza
Ulaz u proceduru
Geografija
Istorija
Kultura
1. Korak
Lokacija
Mentalitet
2. Korak
Razvijenost
3. Korak
Populacija
Društveno uređenje
4. Korak
Vojska
Arhitektura
Druga faza
Generisanje puteva i zgrada
Mapa

Slika 2.1

3 GENERISANJE KARAKTERISTIKA NASELJA

Želja je da se dizajnira fleksibilan algoritam koji bi mogao da se koristi u različitim igrama gde bi se na različite načine tumačio pojam naselja.

Pre svega moramo definisati šta je to naselje, šta su to karakteristike naselja i kako ćemo ih predstaviti. Cilj je da bude što intuitivnije, ali istovremeno i lako za sračunavanje i manipulaciju.

Selo će biti predstavljeno kao apstrakcija koja može da sadrži skup karakteristika. Ideja je da se može definisati proizvoljan broj tipova karakteristika (nezavisno od samog naselja), uvideti zavisnosti između njih i na koji način zavise tj. definisati formule zavisnosti. Jednom definisana karakteristika se može sačuvati i koristiti kasnije po potrebi. Npr. u RPG igri bi vojska najverovatnije predstavljala nebitnu karakteristiku naselja, ali u RTS igri bi bila neizostavna, dok bi za generisanje zadataka za igrača važilo obrnuto. Na slici 2.1 je data jedna od mogućih opcija koja će u daljem izlaganju biti korišćena kao primer. Same karakteristike kao i veze između njih će biti izložene detaljnije u nastavku.

Radi intuitivnijeg objašnjenja karakteristike ćemo zamisliti kao strukture koje sadrže polja (uglavnom brojevnih tipova) i listu zavisnosti. Način izračunavanja svakog polja će biti izložen za svaku karakteristiku. Kod nezavisnih parametara imamo samo polja.

Implementacija karakteristike i naselja se može ostvariti na različite načine zavisno od koncepta u kojem se programira. Jedna od mogućih bi bila objektno-orijentisana implementacija gde bi karakteristika bila apstraktna klasa koja bi sadržala listu drugih karakteristika od kojih zavise i apstraktne metode za formule zavisnosti. Izvođenjem bi se definisale različite karakteristike, njihova polja i metode za sračunavanje vrednosti polja na osnovu zavisnosti. Naselje bi moglo da bude klasa koja bi sadržala listu svih karakteristika. Slična ideja se može primeniti i na endžine bazirane na komponentama gde bi se karakteristike kao komponente „kačile“ na naselja.

3.1 ULAZ U PROCEDURU (NEZAVISNI PARAMETRI)

3.1.1 Geografija

Polja:

- voda/kopno
- reljef
- resursi
- obradivo zemljište
- vegetacija

Parametre ovakvog tipa bi najbolje bilo proslediti u vidu slika gde bi jedan teksel predstavljao neki vid informacije. Npr. crno-bela slika odnosa vode i kopna, ili RGB slika gde bi crvena boja predstavljala pustinjsku oblast, zelena šume, a plava otvoren prostor sa vegetacijom (poput livada). Dodatno se može i generisati slika sa odnosom prohodne i neprohodne površine, koja bi olakšala posao algoritmu za generisanje puteva. Eventualno bi resurse možda bilo optimalnije zapamtiti standardno, kao neku listu ili niz lokacija i tipa resursa.

3.1.2 Kultura

Polja:

- Dogmatizam/Skepticizam
- Duhovnost/Materijalnost
- Arhitektura

Dogmatizam/Skepticizam bi najbolje bilo predstaviti kao realan broj u intervalu [0, 1] koji bi govorio na koju stranu naginje konkretna kultura. Polje *Dogmatizam/Skepticizam* bi se moglo koristiti kada se bude sračunavalo koliko će stanovnici biti obrazovani, kakav će biti njihov tehnološki razvoj i umetnost. Na primer, u srednjem veku je vladala potpuna stagnacija nauke i umetnosti pod uticajem crkvene dogme. *Duhovnost/Materijalnost* se može koristiti pri odlukama koliko će dato naselje pomagati drugima u vidu resursa. Duhovnost takođe može biti put da se otključaju neke „moći“ koje bi stanovnici posedovali.

Pojam arhitekture u kontekstu ovog algoritma će biti razrađen kasnije kada se bude opisivao rad algoritma za generisanje geometrije. Za sada imajmo u vidu da on predstavlja samo neki skup parametara i oblika za taj algoritam.

3.1.3 Istorija

Polja:

- Ratovi
- Katastrofe
- Prethodno naselje

Ratovi i katastrofe bi se mogli jednostavno predstaviti kao brojevi koji bi predstavljali njihovu „jačinu“ u usvojenim jedinicama.

Prethodno naselje bi moglo biti ili referenca na neko naselje koje neće biti postavljeno u igru, ili podskup njenih relevantnih karakteristika. Koristilo bi se pri izračunavanju nekih karakteristika koja naselje može poprimiti od prethodnog (npr. arhitektura, tehnologija itd.).

3.2 KARAKTERISTIKE NASELJA (ZAVISNI PARAMETRI)

3.2.1 Lokacija

Zavisnosti:

- Geografija (sva polja)

Polja:

- Koordinate
- Dostupnost vode
- Resursi
- Obradivo zemljište
- Vegetacija
- Promet

Sve veličine bi se mogle predstaviti kao brojevi koji bi predstavljali njihovu „količinu“ u nekim usvojenim jedinicama. Prirodno, koordinate bi bile koordinate terena. Resursi bi eventualno sadržali broj za svaki tip resursa.

Izračunavanje:

Poziciju naselja, u smislu koordinata na terenu, bi trebalo izabrati smisleno tj. što bliže vodi, resursima i zdravom zemljištu. Algoritam koji ćemo koristiti za to jeste iterativna realizacija Veberovog problema za izbor optimalne lokacije između datih tačaka. On kao ulaz prima tačke koje su nam od interesa i njihove pridružene težinske vrednosti, a zatim kreće od neke inicijalne tačke i iterativno je pomera po onoj osi gde će težinska suma distanci date tačke od ulaznih biti manja od prethodne. To ponavlja dok god ne dođe do tačke odakle ni u jednom smeru ne može da dobije manju težinsku sumu distanci.

Algoritam se može optimizovati dobrim izborom inicijalne tačke aproksimacije (npr. aritmetička sredina proizvoda svih ulaznih tačaka i njihovih težina) što bi generalno trebalo da smanji broj iteracija do krajnje tačke. Takođe se korak pomeraja po osama može povećati pošto nam nije zaista toliko bitna preciznost optimalne lokacije, a može se i uvesti maksimalni dozvoljen broj iteracija čime bi se moglo izračunati vreme rada algoritma u najgorem slučaju. Time bismo uz dobru kalkulaciju koraka pomeraja i maksimalnog broja iteracija mogli da obezbedimo da algoritam uvek da „optimalnu“ lokaciju za neko željeno vreme.

Ulazne tačke bi predstavljale koordinate resursa, vode itd., a težine bi trebalo pridružiti smisleno (npr. voda bi trebalo da nosi najveću težinu). Najbolje je da se parametrizuju radi bolje kontrole algoritma.

Nakon što odredimo koordinate naselja, onda se ostala polja (osim prometa) mogu jednostavno izračunati obrnuto srazmerno udaljenosti od sela i srazmerno njihovoj količini. Informacije poput obradivog zemljišta i vegetacije se mogu jednostavno uzorkovati sa slike po nekom prostoru oko koordinata sela (npr. nekog zadatog poluprečnika).

Sračunavanje prometa stvara mali problem u smislu odstupanja od glavne ideje podele na korake jer bi se mogao realno sračunati tek nakon što se generišu sva naselja i bude imala informacija o putevima između njih.

3.2.2 Arhitektura

Zavisnosti:

- Kultura (arhitektura)
- Istorija (prethodno naselje (arhitektura))

Polja:

- Elementi arhitekture (oblici)
- Tipovi ulica

Detaljnije će biti izloženo nakon opisa rada algoritma za generisanje geometrije (Generisanje geometrije).

3.2.3 Mentalitet populacije

Zavisnosti:

- Istorija (Ratovi)
- Kultura (Duhovnost/Materijalnost)

Polja:

- Agresivan/Miroljubiv
- Pomažući/Sebičan
- Prijateljski/Neprijateljski

Najbolje ih je predstaviti realnim brojevima u intervalu [0, 1].

Izračunavanje:

Polje *Agresivan/Miroljubiv* bi moglo da se izračuna na osnovu broja ratova koji su se dešavali u tom naselju. Narod koji je često ratovao je najverovatnije agresivan. Koristilo bi se npr. kod odluke da li bi dato naselje napadalo susede.

Polje *Pomažući/Sebičan* se može izračunati na osnovu polja *Duhovnost/Materijalnost.* Materijalni ljudi će želeti sve za sebe i neće deliti ništa sa drugima. To bi se moglo koristi kod odluke da li će dato selo donirati neke resurse drugima kada im je to potrebno.

Što se polja *Prijateljski/Neprijateljski* tiče, teško je odrediti realnu zavisnost koju bi imalo smisla ubacivati u ovaj sistem tako da bi najbolje bilo da se generiše kao slučajan broj u intervalu [0, 1].

3.2.4 Razvijenost

Zavisnosti:

- Lokacija (resursi, vegetacija, obradivo zemljište, promet)
- Kultura (Dogmatizam/Skepticizam)

Polja:

- Industrija
- Trgovina
- Poljoprivreda
- Stočarstvo
- Nauka/Tehnologija/Obrazovanje

Izračunavanje:

Ova polja bi trebalo implementirati kao brojeve koji bi predstavljali njihovu „jačinu“ u usvojenim jedinicama.

Svako polje bi se računalo određenom formulom zavisnosti u odnosu na neku od navedenih zavisnosti. Industrija bi zavisila od prisutnosti resursa u regionu, trgovina od prometa, poljoprivreda od obradivog zemljišta, stočarstvo od vegetacija, a naučno-tehnološki razvoj i obrazovanje od toga da li ljudi razmišljaju dovoljno skeptično.

3.2.5 Populacija

Zavisnosti:

- Razvijenost (funkcija svih grana)
- Istorija (ratovi, prirodne katastrofe)

Polja:

- Broj stanovnika

Izračunavanje:

Broj populacije predstavlja funkciju zavisnosti od razvijenosti i istorijskih događaja u datom naselju. Srazmerno je razvijenosti, a obrnuto srazmerno istorijskim dešavanjima (ratovima i prirodnim katastrofama). Takođe može se dodati i odnos muške i ženske populacije gde bi preovladavala ženska u slučaju velikog broja ratova.

3.2.6 Društveno uređenje

Zavisnosti:

- Razvijenost
- Mentalitet

Polja:

- Uticaj na vojsku
- Tip uređenja

Izračunavanje:

Tip uređenja ćemo u ovom slučaju podeliti na četiri tipa:

1. Diktatura,
2. Demokratija (Republika)
3. Aristokratsko uređenje,
4. Imperija.

Svaki od datih tipova na različit način utiče na stanovništvo i vojsku i svako se javlja u različitim uslovima.

Diktatura nastaje ako je razvoj na niskom nivou, demokratija ako je obrazovan, a aristokratsko uređenje (vladavina više klase) ako je bogat (razvoj je na visokom nivou). Za imperiju je potrebno malo više preduslova, a to su da narod bude agresivan, razvijen (u smislu privrede) i obrazovan. Potrebno je izračunati šansu za razvoj svakog od uređenja i onda izabrati onaj sa najvećom vrednošću. Opciono se može dodati i slučajno odstupanje kako bi algoritam bio manje deterministički.

Shodno uređenju bi postojalo više ili manje vojske. Npr. imperija bi najviše forsirala militaristički razvoj kako bi mogla stalno da se proširuje. Takođe se može uvesti i veličina koja bi predstavljala koliko je narod zadovoljan, na šta bi uticalo društveno uređenje i to bi se manifestovalo češćim ili ređim bunama. Bune bi se mogle predstaviti kao smanjenje broja stanovnika sa nekom šansom da se promeni uređenje. Neka polja bi se morala rekalkulisati u tom slučaju.

3.2.7 Vojska

Zavisnosti:

- Populacija (Broj)
- Razvijenost (Nauka/Tehnologija)
- Društveno uređenje (Uticaj na vojsku)

Polja:

- Broj vojnika
- Moć oružja

Izračunavanje:

Količina vojske bi direktno zavisila od populacije i uređenja. Što veća populacija i što agresivnija politika to veća vojska. Moć vojske, predstavljena kao neki broj u usvojenim jedinicama, bi zavisila od tehnološkog razvoja. Napredniji narod će imati i tehnološki naprednije i moćnije oružje.

3.3 FORMULE ZAVISNOSTI

Konkretne formule zavisnosti mogu biti bilo koje matematičke funkcije dok god poštuju srazmernost. Najjednostavnije, mogu se predstaviti kao količnik srazmernih i obrnuto srazmernih veličina pomnoženih sa koeficijentima koji bi se mogli parametrizovati radi kontrole algoritma. Jednom kad se napiše algoritam može se pokrenuti niz eksperimenata sa različitim koeficijentima sa ciljem da se dobije najbalansiraniji izlaz.

Malo napredniji pristup za određivanje funkcija bi bio da sami izračunamo koje bismo vrednosti želeli za svaku od njih i da onda nekom od metoda interpolacije dobijemo analitičke oblike tih funkcija.

3.4 ZAKLJUČAK

Ovom tačkom završavamo generisanje karakteristika naselja. Moć ovakvog pristupa leži u tome što se jednom osmišljena i definisana karakteristika ne mora ponovo definisati i može se koristiti po potrebi. Ovakvim pristupom možemo lako da aproksimiramo zavisnosti u stvarnom svetu dovoljno precizno za potrebe igara i da ih promenimo kada god to želimo.

Jasno je da se može povući još mnogo zavisnosti, npr. mentalitet zavisi od razvijenosti, arhitektura takođe itd., ali trebalo bi naći kompromis između realnosti i jednostavnosti.

4 GENERISANJE GEOMETRIJE

Nakon što smo definisali naselje i sve njegove karakteristike ostaje još da mu damo fizički smisao. Međutim, pre nego što se počne sa generisanjem geometrije potrebno je prvo da poznajemo gustinu populacije cele mape. Zbog toga je potrebno prethodni algoritam pokrenuti onoliko puta koliko naselja budemo hteli da generišemo i u hodu ucrtavati u mapu populacije potrebne informacije. Tu mapu populacije ćemo kasnije koristiti u algoritmu za generisanje puteva. Nakon što se generišu svi putevi, unutar zona sa skoncentrisanom populacijom (lokacija naselja) blokovi između puteva se dele na manje parcele koji predstavljaju lokacije na kojima će se graditi zgrade. Zatim se te lokacije prosleđuju algoritmu za generisanje zgrada koji koristi informacije o arhitekturi naselja kako bi proceduralno izgradio sve potrebne zgrade. U nastavku prvo će biti objašnjen algoritam za generisanje puteva, a nakon toga algoritam za generisanje zgrada.

4.1 GENERISANJE PUTEVA

4.1.1 Uvod

Mrežu puteva ćemo predstaviti kao neusmereni graf, gde bi svakom čvoru unutar mreže puteva odgovarao jedan čvor u grafu. Biće implementiran preko lista susednosti, pošto su operacije sa susedima česte.

Postoje dve vrste puteva: ulice i autoputevi. Autoputevi će služiti da povežu različita naselja, a ulice da formiraju mrežu puteva unutar jednog naselja. Prvo se pokreće jedan prolaz za generisanje autoputeva, a zatim za generisanje ulica za svako naselje.

Algoritam radi tako što opslužuje zahteve za generisanje segmenta puta i nakon svakog uspešno opsluženog generiše određen broj novih zahteva. Segment puta je predstavljen kao grana između dva čvora.

4.1.2 Algoritam

Zahtevi za generisanje segmenata puteva su predstavljeni u obliku **req(timeDelay, node, metaInfo)**. Parametar *timeDelay* predstavlja zakašnjenje nakon kojeg će zahtev biti obrađen, *roadNode* čvor koji želimo da dodamo u graf, a *metaInfo* informacije koje su relevantne za funkcije koje koristi algoritam.

Postoje dve takve funkcije:

- funkcija *gloalGoals()* (skraćeno **GG**) za kreiranje zahteva na osnovu globalnih ciljeva,
- funkcija *localConstraints*() (skraćeno **LC**) za obradu zahteva na osnovu lokalnih ograničenja.

Funkcija **GG** kreira nove zahteve, a funkcija **LC** ih obrađuje i ako budu uspešno obrađeni ubacuje ih u graf mreže puteva. Pseudo kod i objašnjenje su dati u nastavku.

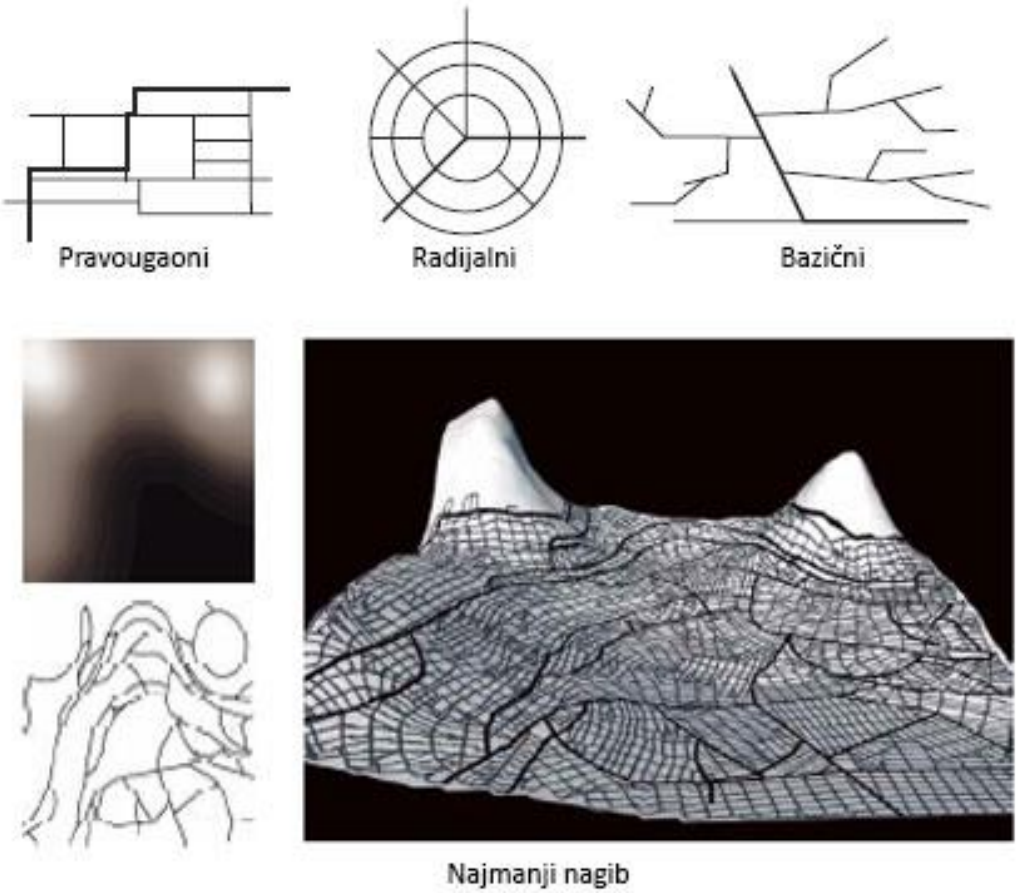
Zahtevi se ređaju u prioritetni red **Q**. Inicijalizuje se sa prvim segmentom puta. **LC** funkcija ga obrađuje, modifikuje po potrebi i vraća status da li je zahtev uspešno obrađen ili ne. Ako je zahtev uspešno obrađen čvor se ubacuje u graf mreže puteva **S** i poziva se **GG** funkcija koja generiše nove zahteve čime se, zapravo, realizuje grananje tog poslednjeg segmenta puta. Ti zahtevi se dodaju u red zahteva **Q**. Ceo postupak ponavljamo dok god se red ne isprazni čime smo završili generisanje grafa.

```
initialize priority queue Q with a single entry: r(0, node0, metaInfo0)
initialize segment graph S to empty

until Q is empty
  delete smallest r(timeDelay_i, node_i, metaInfo_i) from Q (i.e., smallest 'timeDelay')
  accepted = localConstraints(&r)
  if (accepted)
  {
    add segment(node_i) to S
    foreach r(timeDelay_j, node_j, metaInfo_j) produced by globalGoals(node_i, metaInfo_i)
      add r(timeDelay_i + 1 + timeDelay_j, node_j, metaInfo_j) to Q
  }
```

Funkcija GG može kreirati grananja na više različitih načina što specificiramo parametrom *metaInfo*. Funkciju treba dizajnirati tako da bude lako proširiva za nove tipove puteva. Ovde će biti prikazano nekoliko najčešćih tipova tipova. Za autoputeve će postojati samo jedan, a za ulice ćemo koristiti četiri tipa (slika 4.1):

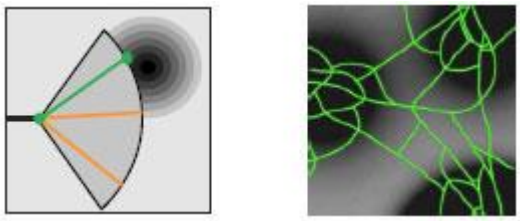
1. pravougaoni,
2. radijalni,
3. bazični,
4. najmanji nagib.



Slika 4.1

Svaki od tipova bi koristio različite parametre. Npr. radijalni bi koristio polarni ugao, bazični maksimalni ugao skretanja itd. Željeni tip puta možemo postavljati na osnovu arhitekture naselja, što nas dovodi do prvog polja za karakteristiku naselja Izgled (Arhitektura). Sama propagacija puta se realizuje konceptom „kornjače“ tj. pokretnog kursora koji se uvek pomera relativno. Kod pravougaonih puteva bi se uvek rotirala za 90 stepeni, kod radijalnih za neki polarni ugao itd.

Autoputevi treba da povezuju više naselja. Slika populacije će biti korišćena kako bi se usmerila propagacija autoputeva. Smer u kome treba da nastavi autoput može se odrediti tako što bi se „pucali“ zraci (poluprave) u nekoliko smerova i sračunavala vrednost srazmerna količini populacije i obrnuto srazmerna udaljenosti od čvora. Tako sumirane vrednosti duž zrakova se upoređuju i ona sa najvećom vrednošću se bira kao smer u kome se nastavlja propagacija autoputa (slika 4.2). Ako su neke vrednosti dovoljno približne treba razgranati put tako da ide ka svakoj od korespodentnih lokacija. Ako se trenutni čvor nalazi unutar prostora sa visokom gustinom populacije, to implicira da je put zašao u naselje i treba ga razgranati standardno dok se ne izađe iz naselja. Takođe se može, radi živopisnosti, sa nekom šansom bezuslovno generisati grana u nekom smeru. Takva grana će ili da završi u nekom od naselja ili da se spoji sa ostatkom mreže puteva formirajući raskrsnicu.



Slika 4.2

Funkcija **LC** vrši obradu jednog zahteva za segment puta i vraća status da li je uspešno obrađen. U nastavku je dato nekoliko primera obrade:

1. Ako čvor ulice upadne nevalidnu zonu vrši se jedna od sledećih akcija:
   1.1. Skraćuje se tako da ne ulazi u nevalidnu zonu
   1.2. Rotira se tako da ide uz ivicu (putevi u obalu)
   1.3. Ako čvor pripada autoputu može se ostaviti na tom mestu i posebno obeležiti, pa kasnije transformisati npr. u most ili tunel
2. Provera sopstvenog stanja:
   2.1. Ako se segment seče sa već postojećim segmentom onda se formira se raskrsnica
   2.2. Ako je čvor blizu već postojeće raskrsnice produži se do raskrsnice
   2.3. Ako je čvor blizu već postojećeg segmenta onda se produžava tako da se formira presek
3. Ako je neuspešna obrada vraća se *false* status

Kao što se da videti, ovakav algoritam je „svestan samog sebe“ i formira petlje/raskrsnice ukoliko je novi čvor dovoljno blizu nekog drugog segmenta ili čvora.

Nakon što se generiše čitav graf može se produžiti dalje sa generisanjem građevina i proslediti sistemu za renderovanje. Između čvorova se, po potrebi, mogu interpolirati krive kako bi se dobio realniji izgled puteva.

4.1.3 Optimizacije

Algoritam se može ubrzati paralelizacijom posla na više niti što bi ga znatno ubrzalo. Tekuća nit može produžiti jednom od novonastalih grana, a kreirati se nova nit za svaku od preostalih (bazen niti bi još više doprineo brzini). Potrebno je sinhronizovati pristup upisa u red **Q** i graf **S**.

Mreža puteva generisana na ovakav način omogućava njenu upotrebu na različitim mestima. Sa lakoćom se mogu na njoj primeniti *pathfinding* algoritmi, iscrtati mapa puteva i, u suštini, primeniti bilo koji algoritam koji radi sa grafovima.

4.2 GENERISANJE GRAĐEVINA

4.2.1 Uvod

U svrhu proceduralnog generisanja građevina ćemo koristiti koncept formalne gramatike koja se naziva gramatika deljenja (*Split grammar*) i bazirana je na gramatici oblika (*Shape grammar*) čije je istraživanje započeo Stiny[1]. Algoritam operiše nad oblicima i kao i svaka formalna gramatika sastoji se iz minimum tri skupa (slika 4.3):

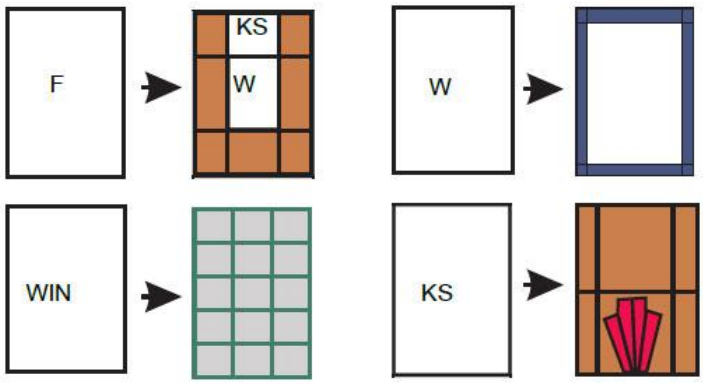
- V: skup promenljivih
- W: skup aksioma (inicijalno stanje)
- P: skup pravila preslikavanja

U skup promenljivih ulaze svi osnovni oblici koje bismo koristili za generisanje neke građevine (kocka, sfera, prozor, vrata, krov itd.). Teorijski svaki oblik se može sintetisati iz primitivnih oblika (kocka, sfera itd.), ali takav pristup je krajnje nepraktičan. Ideja je da se primitivnim oblicima generiše samo „gruba slika“ građevine na koju bi se kasnije dodavale boje, fasade, prozori, vrata, dimnjaci, terase i ostali elementi neke arhitekture. Na taj način smo uz malu žrtvu proceduralne logike dobili jako fleksibilan i jednostavan algoritam koji može da sagradi veliki broj najrazličitijih građevina. Takođe se više arhitektura može izmešati u jednu uzimajući po nešto iz svačijeg skupa promenljivih, što se idealno uklapa u potrebe algoritma i ideje da arhitektura naselja zavisi od istorije i naselja koje je tu bilo pre njega.

Skup aksioma predstavlja inicijalno stanje od koga se započinje preslikavanje.

Skup pravila predstavlja sva preslikavanja koja su moguća u datom sistemu.

Skup aksioma

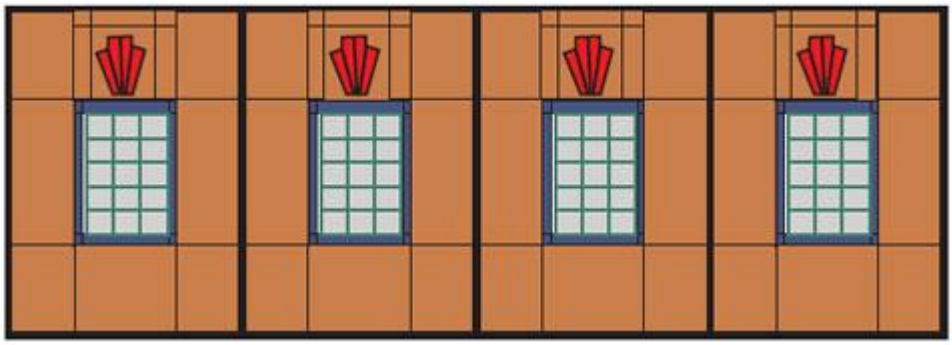


Skup pravila preslikavanja

Slika 4.3

[1] George Stiny je Američki teoretičar dizajna i kompjutera. On je jedan od kreatora koncepta gramatike oblika.

Nakon što se pročita inicijalni skup, počinje se sa iterativnim procesom transformacija koristeći pravila data u skupu pravila preslikavanja. Za primer sa slike 4.3, dobija izlaz na slici 4.4.

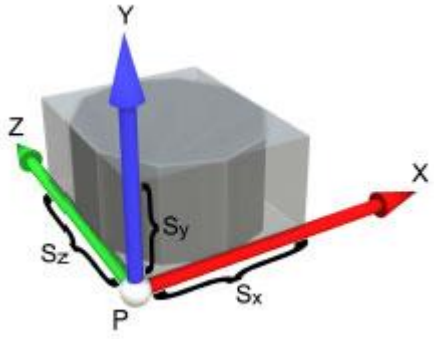


Finalni izlaz

Slika 4.4

4.2.2 Algoritam

Gramatika oblika radi nad oblicima koji se sastoje od: simbola (stringa), geometrije (geometrijskih atributa) i numeričkih atributa. Simboli mogu biti terminalni i neterminalni, a shodno tome se korespodentni oblici nazivaju terminalni i neterminalni. Proces preslikavanja se zaustavlja kada se naiđe na terminalni simbol. Najvažniji geometrijski atributi su pozicija *P*, ortogonalni vektori *X*, *Y* i *Z* koji opisuju koordinatni sistem oblika i vektor veličine (skaliranja) *S*. Ovi atributi opisuju ivičnu kutiju (*bounding box*) zvanu *scope* (slika 4.5).



Ivična kutija (*scope*)

Slika 4.5

Proces preslikavanja funkcioniše u tri koraka:

1. Selektuj aktivni oblik sa simbolom B
2. Izaberi pravilo preslikavanja za simbol B da bi izračunao njegovog sledbenika BNEW
3. Obeleži B kao neaktivan i dodaj BNEW oblike u novu konfiguraciju i vrati se na korak 1.

Kada više nema neterminalnih simbola proces se završava.

Pravila se zadaju u formi *id : prethodnik : uslov -> sledbenik : verovatnoća*

Id predstavlja jedinstveni celobrojni identifikator pravila preslikavanja, *prethodnik* simbol koji treba biti zamenjen simbolom *sledbenik, uslov* logički izraz koji mora biti ispunjen da bi se pravilo primenilo, a *verovatnoća* vrednost u intervalu [0, 1] koja definiše šansu da se pravilo izvrši.

U nastavku će biti date još neke funkcionalnosti koje je potrebno implementirati u algoritam radi lakšeg i efikasnijeg pisanja pravila.

Transformacije i instanciranje nekog modela/primitive:

Primer: A -> [T(0, 0, 6) Rx(90) I("window.fbx")]

T, Rx, Ry, Rz i *S* predstavljaju translaciju, rotacije oko osa Dekartovog koordinatnog sistema i skaliranje respektivno, *I* operaciju instanciranja nekog modela, a srednje zagrade [] predstavljaju simbole za čuvanje i restauriranje trenutnog koordinatnog sistema.

Podela:

Primer: fac -> Subdiv("Y", 3.5, 0.3, 3, 3, 3) { floor, ledge, floor, floor, floor }

Pod navodnicima se specificira osa po kojoj se vrši podela, u nastavku dužine svakog podeoka, a u vitičastim zagradama simboli koji odgovaraju datim podeocima. Korisno je i implementirati specificiranje relativnih dimenzija jer često ne znamo dimenzije oblika kojeg želimo da podelimo.

Ponavljanje:

Primer: floor -> Repeat("X", 2) { B }

Simbol B će biti generisan na svake dve jedinice duž X ose na dužini ivične kutije.

Podela na komponente:

Primeri:

fac -> Comp("faces") { A }

fac -> Comp("edges") { B }

fac -> Comp("vertices") { C }

Služi za podelu nekog oblika sa n dimenzija u oblik sa manje dimenzija. Npr. često nam je potrebno da iz kvadra izdvojimo strane kao fasade neke zgrade. Posebno za tu namenu, pošto je jako česta, zgodno je implementirati i opciju *Comp("side faces")*.

Same oblike možemo implementirati strukturom *octree*. Ona se nameće kao jako dobro rešenje zbog čestih modifikacija i podela oblika.

Može se ići dalje sa algoritmom i da se implementiraju funkcionalnosti za proveru preseka dva oblika kako bi se izbegle neke neželjene situacije (generisanje prozora koji delom upada u stub koji viri iz zida), ili proveru preseka linije vidljivosti jednog oblika sa drugim oblikom što bi bilo korisno generisanje

vrata ako dati zid vidi ulicu. Takođe je jako korisno implementirati opciju za *„prikopčavanje“ (snap),* kako bi se npr. dobili pravilno raspoređeni prozori na zgradi.

Ovakav algoritam daje jako dobre i efikasne rezultate. Za relativno mali skup pravila (u proseku 15) i bazu elementarnih oblika mogu se proceduralno generisati najrazličitije zgrade. Primer je dat u nastavku (slika 4.6).

Slika 4.6

Ovakav algoritam takođe može da se koristi i na ostale komponente urbane okoline. Sledeći primer pokazuje takvu primenu (slika 4.7).

Slika 4.7

Gramatika bi u ovom slučaju uopšteno koristila sledeću strategiju:

1. Podeli ivice placa podelom na komponente i postavi žbunje blizu ograde
2. Podeli plac na prednji, zadnji i glavnu zgradu
3. Generiši stazu i postavi drveće
4. Generiši kuću sa nekim od već gotovih pravila (pretpostavka je da do ovog momenta već imamo napisan bar jedan skup pravila za generisanje kuće)

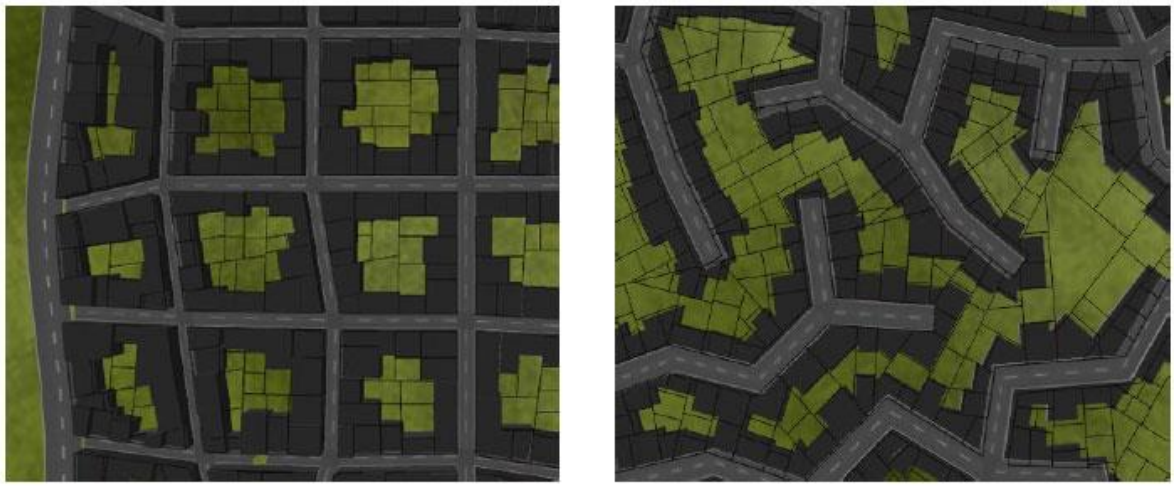
Drveće se može veoma efikasno generisati algoritmima baziranim na L-sistemima, koji funkcionišu jako slično kao i algoritmi bazirani na gramatici oblika.

4.2.3 Veza sa mrežom puteva i karakteristikama naselja

Nakon što smo dizajnirali algoritam za generisanje zgrada sve što je ostalo jeste da iz grafa mreže puteva pokupimo lokacije na kojima je potrebno graditi zgrade i iz karakteristike (Arhitektura) pokupimo skup elementarnih oblika od kojih će te zgrade biti sagrađene.

Time smo i okvirno definisali karakteristiku naselja (Arhitektura), što smo bili odložili do ovog trenutka. Dakle ona se sastoji od tipa ulica, elemenata arhitekture zgrada, a može da sadrži i razne druge parametre (npr. prosečan broj spratova).

Da bismo dobili lokacije na kojima će se generisati zgrade prvo je potrebno izdeliti mrežu puteva na blokove. To se može postići David Eberly-jevim algoritmom za pronalaženje minimalnog ciklusa u grafu u čije razmatranje sada nećemo ulaziti. Kada pronađemo takav ciklus pamtimo ga u listu blokova i izbacujemo iz grafa i ponavljamo taj postupak sve dok graf ne postane acikličan ili mu se ne iscrpe svi čvorovi. Zatim se svi blokovi rekurzivno dele na manje parcele dok se ne dostigne neka minimalna veličine. Pamte se samo one lokacije koje su uz ulicu, a ostale se zanemaruju (slika 4.8).



Slika 4.8

Prazni regioni okruženi zgradama se mogu iskoristi za parkove, parkinge, magične toteme, portale i bilo šta drugo što nam padne na pamet.

4.2.4 Optimizacije i dodatne ideje

Pošto sve zgrade u neku ruku prate određen šablon, bez obzira na arhitekturu (sve imaju vrata vrata, prozore, jedan ili više spratova itd.), mogu se napisati predefinisana pravila za generisanje zgrade koja bi bila parametrizovana čime bi se postiglo na fleksibilnosti. Moglo bi i dosta logike da se pomeri u eksterne funkcije koje bi bile pozivane od strane pravila preslikavanja. Na taj način bi potrebe za promenom skupa pravila preslikavanja bila minimalna i sva bitna logika bi se radila u eksternim funkcijama što je daleko fleksibilnije. Slična stvar je urađena u L-System algoritmu za generisanje puteva koje su razvili Parish i Müller.

S obzirom na to da postoji preslikavanje jedan na jedan zgrade na parcelu i ne postoje nikakve izmene deljenih podataka postoji veliki potencijal za paralelizaciju. Za razliku od algoritma za generisanje puteva gde se vrše česte izmene deljenog resursa, ovde takva situacija ne postoji pa se može postići masovna paralelizacija tehnikama GPGPU programiranja. Time bi se postigao veliki skok u brzini rada algoritma.

5 ZAKLJUČAK

Ovakav algoritam pruža veliku fleksibilnost pri generisanju naselja sa mogućnošću najrazličitijih izlaza. Dizajniran je tako da se može lako proširivati novim mogućnostima i samim tim bi trebalo da evoluira zajedno sa igrama u kojima bi se koristio. Postoji još mnogo prostora za unapređenje. Jedna od ideja bi bila da se proširi mogućnošću za proceduralno generisanje biljaka na njivama u naseljima (npr. primenom L-sistema).

6 LITERATURA

P. Müller, P. Wonka., S. Haegler, A. Ulmer, L. V. Gool – *Procedural Modeling of Buildings*

Y.Parish, P. Müller – *Procedural Modeling of Cities*

G. Kelly, H. McCabe – *Citygen: An Interactive System for Procedural City Generation*

G. Kelly, H. McCabe – *A Survey of Procedural Techniques for City Generation*

Jingyuan Huang, Alex Pytel, Cherry Zhang, Stephen Mann, Elodie Fourquet, Marshall Hahn, Kate Kinnear, Michael Lam, and William Cowan, David R. – *An Evaluation of Shape/Split Grammars for Architecture*

Bjorn Gunnar Eilertsen – *Automatic road network generation with L-systems and genetic algorithms*

Nemanja Đurkić – *Diplomski rad* JAVA APLIKACIJA ZA VIZUELNU REPREZENTACIJU WEBEROVIH MULTIFOKALNIH KRIVIH U METRICI MINKOWSKOG