Vladimir Makarić

Poređenje efikasnosti simulacija grupnog kretanja

Grupno kretanje predstavlja kretanje entiteta pri kojem se svi entiteti istovremeno kreću ka zajedničkom cilju. Entitet je apstraktni pojam koji predstavlja mnoge učesnike grupnog kretanja, na primer: pticu u jatu, antilopu u krdu, tenk na bojnom polju itd. Cilj ovog projekta je poređenje efikasnosti algoritama koji simuliraju grupno kretanje. Entiteti su u simulaciji predstavljeni kao materijalne tačke sa određenim minimalnim međusobnim rastojanjem. Kreću se u dvodimenzionalnom prostoru, sa definisanom polaznom i završnom tačkom, u kojem dodatno mogu postojati prepreke. Prvi algoritam se bazira na ideji da svi entiteti prate jednog koji je označen kao vođa grupe, drugi u kome svaki entitet prati sebi najbliži entitet i treći u kome se svaki entitet kreće samostalno. Algoritmi su poređeni po sledećim kriterijumima: ukupna pređena distanca i ukupno vreme kretanja svih entiteta. Efikasnijim se smatra algoritam kod koga su oba merila što manja. Očekivalo se da treći algoritam pokaže najveću efikasnost po ovim kriterijumima. Testiranjem je utvrđeno da je najefikasniji algoritam treći, potom prvi, a najlošije se pokazao drugi.

Uvod

U prirodi se često susrećemo sa grupnim kretanjem. Primeri su jato ptica, krdo antilopa, kolonija mrava itd. Primera ima i kod ponašanja ljudi. Kretanje tenkova na bojnom polju, evakuacija ljudi iz zgrade prilikom požara i sličnim vanrednim situacijama itd. U ovom projektu su učesnici grupnog kretanja nazvani entiteti. Oni su apstrahovani materijalnim tačkama sa određenim minimalnim međusobnim rastojanjem. Prostor u kome se simulacija odvija je dvodimenzionalan sa podesivim preprekama. Prepreke mogu biti konveksni poligoni, ili kombinacija dva ili više konveksnih poligona spojenih tako da je stvoren privid nekonveksnog poligona. Postavljanje prepreka u prostor kojim se entiteti kreću unosi dodatnu kompleksnost u razmatranje algoritma za simulaciju grupnog kretanja.

Vladimir Makarić (1993), Novi Sad, Neimarova 15, učenik 2. razreda Gimnazije „Jovan Jovanović Zmaj" u Novom Sadu

Kretanje entiteta teče samo po x i y osi. Orijentacija (ugao pod kojim je rotiran entitet) nije uzeta u obzir pošto su jedinke predstavljene pomoću materijalnih tački. Iako postoji mnogo različitih načina grupnog kretanja, u ovom radu se poredi efikasnost tri načina kretanja koji su modelovani pomoću tri algoritma. Prvi algoritam određuje vođu grupe koji je jedini entitet koji „zna" put do cilja, a ostali entiteti ga prate. Drugi algoritam se bazira na ideji da svaki entitet prati sebi najbližeg. U trećem algoritmu svaki entitet zna put do cilja. Put do cilja se izračunava pomoću algoritma koji će biti objašnjen kasnije. Efikasnost navedenih algoritama se poredi u odnosu na ukupan pređeni put do cilja i ukupno vreme do stabilizacije (dostizanja ciljnog stanja).

Metod

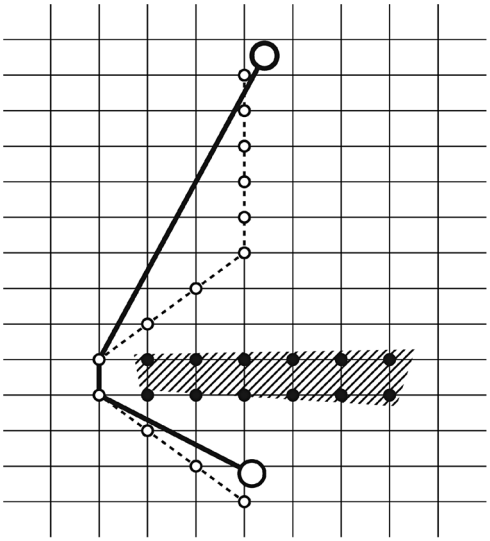
Program je pisan u programskom jeziku C++ a za prikaz simulacije je odabran grafički interfejs iz razloga što je preglednije posmatrati simulaciju vizuelno nego numerički. Za iscrtavanje grafičkih oblika koji predstavljaju elemente simulacije koriste se OpenGL primitive.

Agloritam je iteracioni i svaka iteracija redom ima 4 koraka:

1. Računanje putanje za svaki entitet
2. Računanje kretanja po izračunatoj putanji, to jest određivanje vektora brzine za svaki entitet
3. Algoritmi provere, tj. izbegavanja sudara za svaki entitet
4. Ažuriranje pozicije svih entiteta koji su dobili dozvolu u prošlom (trećem) koraku.

1. Računanje putanje

Kao preduslov za bilo kakvo grupno kretanje, mora postojati način računanja putanje za pojedinačne entitete. Algoritam koji je izabran za gore navedenu svrhu je A* (web 1). Međutim za A* je potrebna matrica ili graf, a prepreke su definisane kao vektorski predstavljeni poligoni (šrafirani četvorougao, slika 1). Prostor problema je aproksimiran dvodimenzionalnom rešetkom čiji čvorovi se označavaju kao dostupni ili nedostupni (mali crni krugovi u poligonu, slika 1) u zavisnosti od toga da li su „prekriveni" poligonima. Putanja koja se isprva dobija je definisana kao uređen niz čvorova iz rešetke (isprekidana linija, slika 1). Potom se putanja pojednostavljuje, tako što se izbacuju svi nepotrebni čvorovi a potom se putanja ponovo prevodi u vektorsku (debela crna linija, slika 1). Time se dobija prirodnije kretanje objekata, nasuprot mnogo diskretnijeg cik-cak kretanja koje bi se odvijalo isključivo po linijama rešetke.



Slika 1.
Računanje putanje

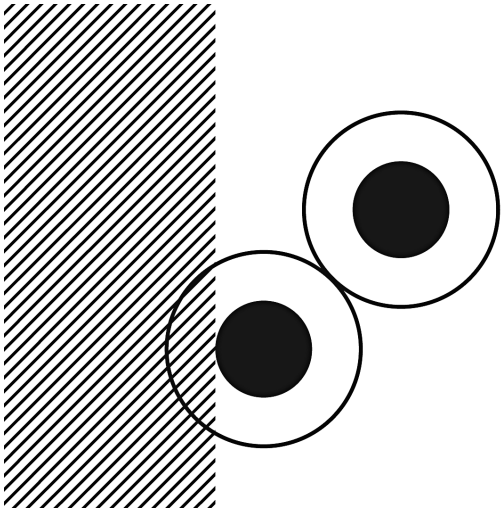
Figure 1.
Path calculation

2. Kretanje pojedinačnog entiteta po putanji

Vektor brzine (konstantnog intenziteta) se računa u odnosu na sledeću tačku u putanji entiteta – to je brzina kretanja. Ta sledeća tačka se označava kao trenutni cilj, i jedinka se kreće do te tačke. Kriterijum za prelaz na sledeću tačku je postizanje minimalnog rastojanja u odnosu na trenutni cilj. Kada se entitet dovoljno približi trenutnom cilju, tada se za trenutni cilj uzima sledeća tačka na putanji. Međutim ponekad se entiteti ne kreću direktno po svojoj putanji, jer bivaju izgurani od strane drugih entiteta. Tada je moguće da entitet nikad ne zadovolji to minimalno rastojanje, to jest moguće je da bude izguran ispred trenutnog cilja. Tada entitet treba da nastavi napred ka sledećoj tački, međutim nije zadovoljeno minimalno rastojanje tako da se on ustvari vraća nazad do svog trenutnog cilja koji je sada iza njega, jer je bio izguran napred. Ovaj problem je rešen tako što se u svakoj iteraciji proverava prolaznost putanje od trenutne pozicije entiteta pa do sledeće tačke u putanji, posle trenutnog cilja.

3. Provera sudara

Oko svakog entiteta postoji zamišljena kružnica, „ljuska" (ivica većeg kruga, slika 2), koja predstavlja minimalno međusobno rastojanje u odnosu na druge entitete. Postoje dve provere sudara, prva je između samih entiteta, to jest njihovih ljuski, a druga je između entiteta i prepreka (šrafirani pravougaonik, slika 2). Druga provera sudara se računa između poligona kojim je predstavljena prepreka i poligona kojim je predstavljen entitet. Preko ljuske se obavlja provera sudara između samih entiteta a



Slika 2.
Provera sudara između entiteta i prepreka

Figure 2.
Collision detection between entities and obstacles

provera u odnosu na prepreke se obavlja preko samog geometrijskog objekta (crni pun krug, slika 2) kojim je entitet predstavljen zanemarujući ljusku.

4. Rešavanje sudara

Pri kretanju više od jednog entiteta potrebno je detektovati sudare. Rešavanje sudara je potrebno jer se pri računanju putanje ostali entiteti ne uzimaju u obzir. Zbog toga je potrebno detektovanje i rešavanje sudara jer može doći i do sudara između entiteta. U svrhu rešavanja sudara razvijena su dva algoritma:

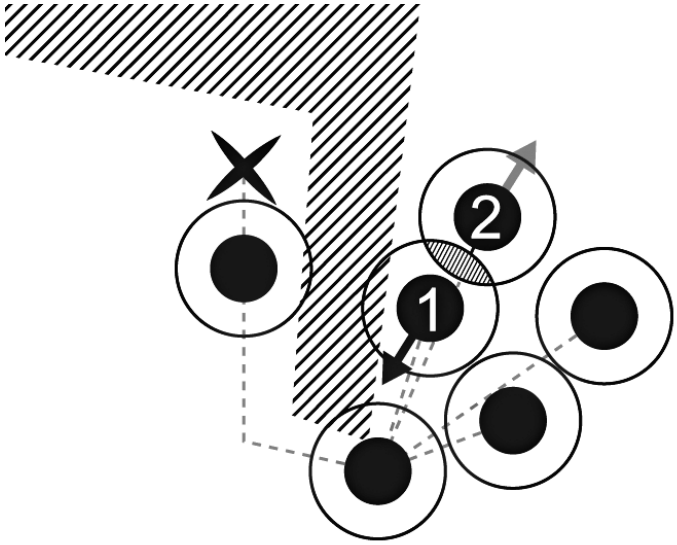
1) **Reakcija na sudar**

Međusobni sudari

Radi na principu kruga (ljuska). Krug je poluprečnika željenog rastojanja. Za vreme kretanja vrši se provera sudara i pri svakom sudaru se jedinke raspoređuju tako da se na njihovu poziciju dodaje vektor pomeraja kojeg određuje provera sudara. Primer je dat na slici 3, na kojoj se vide pozicije dva entiteta tačno jednu iteraciju pre sudara. U sledećoj iteraciji dolazi do sudara, sudar se detektuje (šrafirana površina, slika 3) i izračunava se vektor pomeraja za oba entiteta (slika 3, vektori). Vektor pomeraja se primenjuje na jedan od dva entiteta (slika 3). Na koji entitet će biti primenjen vektor pomeraja, to jest koji entitet će se pomeriti da bi se otklonio sudar, zavisi od hijerarhije reakcije na sudar.

Sudari sa preprekama

Pošto pronalaženje putanje ne uzima u obzir ostale entitete, dešava se da usled međusobnog sudaranja i odbijanja entiteti siđu sa svojih putanja.



Slika 3.
Hijerarhija rešavanja sudara

Figure 3.
Collision solution hierarchy

Silaženje sa putanja može da dovede do sudara sa preprekama. U svrhu provere i reakcije na sudar između entiteta i prepreka implementiran je hibridni SAT algoritam (Separating Axis Theorem) (web 2). Hibridni SAT pored provere sudara, vraća vektor pomeraja koji omogućava reakciju na sudar.

Hijerarhija rešavanja sudara

U slučaju da objekti siđu sa putanje, desi se da udare u prepreku i tu se grupišu iako bi po putanji trebalo da je zaobiđu. Može da se desi da svojim međusobnim odbijanjima oni najbliži zaobilaženju prepreke zapravo guraju one koji su dalji, samim tim celu grupu udaljavajući od cilja. Da bi se izbegle takve situacije, pri proveri sudara određuje se da onaj koji će se pomeriti kao rezultat sudara bude bliži cilju, to jest da njegova putanja bude kraća. što znači da oni koji su dalje, guraju one koji su bliže, što rešava situacije lošeg grupisanja. Na primer, na slici 3 došlo je do sudara između entiteta 1 i entiteta 2. Vektor pomeraja je izračunat za oba entiteta i sad je pitanje koji od ta dva entiteta treba da se pomeri da bi se sudar otklonio. Pošto entitet 2 ima dužu putanju do cilja od entiteta 1, entitet 1 će se pomeriti za svoj vektor pomeraja (crni pun vektor). Ovo može da se gleda i kao da je entitet 2 „gurnuo" entitet 1 bliže cilju.

Loše strane ovog pristupa

Kasnije se ispostavilo da u nekim situacijama kada objekti uđu u neki uzak prostor prethodni postupak ima svoje loše strane. Na primer, entiteti mogu da stignu do cilja koji se nalazi između 3 poligona koji formiraju oblik poput slova „U". Tada dolaze do izražaja loše strane

provere sudara sa krugovima. Problem je što se provera odvija u jednom trenutku između samo dva entiteta to jest njihovih ljuski. Ukoliko je došlo do sudara oni se pomere u nekom pravcu. A u tom pravcu može da se nalazi zid za koji taj entitet u trenutku provere sudara ne zna. Tada entitet uđe u zid. Međutim, provera sudara sa zidom vrati entitet nazad i on ostane u sudaru sa istim entitetom kao i pre pomeranja. Tako dolazi do gubljenja rastojanja između entiteta, tj entiteti počnu da ulaze jedni drugima u ljuske.

2) **Izbegavanje sudara**

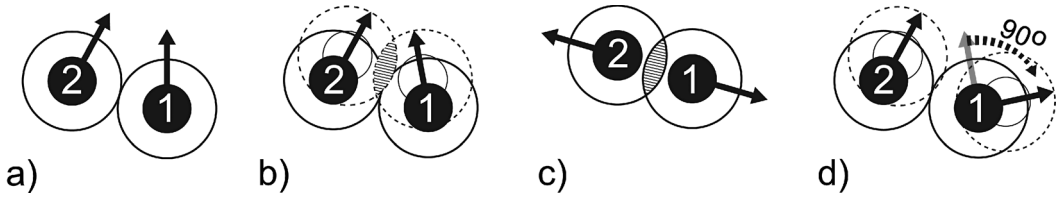
Za razliku od prethodnog metoda koji računa reakciju na sudare, metod izbegavanja sudara „gleda unapred", tj. predviđa i sprečava sudare pre nego što do njih dođe.

Prvi algoritam izbegavanja sudara.

Ulaz algoritma je entitet, primarni entitet. Izlaz je novi smer kretanja to jest nova brzina i dozvola za ažuriranje primarnog entiteta. Ovaj algoritam promene vrši na primarnom entitetu. Svaki entitet u svakom trenutku ima svoj pravac kretanja (koji određuje brzinu kretanja), određen trenutnim ciljem. Pošto je u pitanju grupno kretanje desi se da se entiteti nađu jedan drugom na putu i tada se brzine menjaju. Neka primarni entitet bude entitet broj 1 (slika 4). Vrši se test sudara na ostale entitete, ali tako da se testira buduća pozicija entiteta i buduća pozicija primarnog entiteta, to jest pozicija koja je jednaka trenutnoj poziciji (slika 4a) plus trenutnoj brzini (slika 4b, entiteti sa isprekidanom ljuskom). Ukoliko nema sudara primarni entitet dobija dozvolu da ažurira svoju poziciju svojom brzinom. A ukoliko dođe do sudara, tada se brzina primarnog entiteta mora promeniti da do sudara ne bi došlo kada se pozicije azuriraju. Pošto je brzina vektor, a intenzitet vektora brzine je konstantan, jedini način promene brzine je rotacija vektora. Bilo bi idealno kada bi rotirali vektor brzine za sve moguće uglove. Nažalost to bi bilo previše neefikasno i sporo. Napravljen je kompromis i odlučeno je da su dve alternativne brzine dovoljne za dobar rad algoritma. Ta dve brzine, to jest dva nova smera kretanja (vektori), su vektor brzine kretanja rotiran za 90 stepeni (normalan na vektor brzine) i još jedan za 90 stepeni, samo u suprotnom

Slika 4.
Prvi algoritam izbegavanja sudara izmedu dva entiteta

Figure 4.
The first algorithm for collision avoidance between two entities



smeru. Da li će biti odabran prvi ili drugi zavisi od toga koji od ta dva smera je prošli put odabran za dati entitet i taj će prvi biti testiran. Ukoliko se ne bi pamtio prošli smer kretanja entiteti bi se često nasumično kretali u jednom pravcu i ne bi stigli nigde. U ovom slučaju entitet se kreće jednim pravcem sve dok je on dostupan. Ukoliko je poslednji odabran redovni smer kretanja onda se nasumično određuje koji od dva alternativna smera će biti testiran. Kada se odredi jedan od dva moguća alternativna smera to jest dve alternativne brzine, ta brzina postaje nova brzina. Proveri se sudar sa ostalim entitetima uzimajući u obzir novu brzinu (slika 4d). Ukoliko nova brzina nema intersekcije sa ostalim entitetima ili preprekama, primarni entitet dobija dozvolu da ažurira svoju poziciju ali sa novom brzinom. Ukoliko i ta brzina bude u sudaru sa nečim, onda se proverava druga alternativna brzina. Ukoliko se i to pokaže kao nedostupan pravac kretanja, tada se primarni entitet prosleđuje drugom algoritmu.

Drugi algoritam izbegavanja sudara

Ulaz algoritma je primarni entitet, a izlaz je dozvola za ažuriranje. Ovaj algoritam je znatno složeniji od prvog i koristi se jedino kada prvi algoritam ne uspe da izračuna novu brzinu. Za razliku od prvog algoritma. Ovaj algoritam pokusava da održi brzinu kretanja primarnog entiteta. Tako što pomera entitete (osim vođe ili u slučaju trećeg algoritma, prvog entiteta koji je došao do cilja) sa kojima bi se primarni entitet sudario u sledećoj iteraciji. Ako je to nemoguće, onda dolazi do promene brzine primarnog entiteta. Potom ponovo sledi pomeranje svih entiteta sa kojima bi se primarni entitet sudario u sledećoj iteraciji. Brzine to jest smerovi kretanja primarnog entiteta se menjaju kao i u prvom algoritmu.

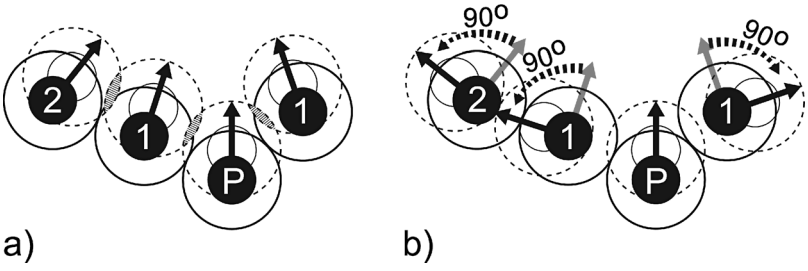
Kao i u prvom algoritmu vrši se provera sudara u odnosu na buduću poziciju primarnog entiteta. Ukoliko se desi sudar sa nekim entitetom, taj smer kretanja to jest ta brzina nije odmah odbačena, nego se nastavlja provera sudara jer je moguće da se detektuje sudar sa 2 ili više entiteta. Svi entiteti sa kojima se utvrdi sudar se označavaju kao deca primarnog identiteta. Kada su ustanovljena sva deca primarnog entiteta, tada se isti algoritam rekurzivno poziva za svu decu. Početni entitet dobija dozvolu za ažuriranje jedino ako sva deca dobiju dozvolu za ažuriranje. U suprotnom se taj smer kretanja odbacuje i proverava se jedan od dva smera kao i u prvom algoritmu. Ukoliko se oba smera pokažu kao zauzeta, onda se proverava suprotan smer od smera kretanja ili prosto rečeno entitet se vraća unazad. Vraćanje unazad nije urađeno rekurzivno nego se proveri da li je slobodan taj pravac, ukoliko nije, entitet nema gde da se pomeri i njemu je ažuriranje zabranjeno. Svi entiteti koji su obrađeni su obeleženi kao obrađeni, jer se može desiti da se zatvori lanac što bi izazvalo beskonačno izvršavanje algoritma. Takođe svaki entitet pamti svog

roditelja ukoliko ga ima, da ne bi svog roditelja dodao kao svoje dete, što bi izazvalo beskonačno izvrsavanje algoritma. Kada se kao deca obeleže entiteti koji su vec obeleženi ili kada se u tom pravcu utvrdi sudar sa preprekom taj pravac se odbacuje. Potom se kreće na drugi pravac, s tim što se pre toga očiste svi entiteti koji su obeleženi kao obrađeni. Svaki put kada jedan pravac kretanja početnog entiteta vrati zabranu ažuriranja, ceo lanac se čisti. Zato što postoji šansa da i drugi mogući pravci kretanja koji će biti testirani imaju zajedničke članove lanca to jest stabla.

Pored izbegavanja sudara ovaj algoritam ima ulogu optimalnog raspoređivanja jedinki u prostoru oko cilja. Pod optimalnim podrazumeva se da u što manji prostor stane što više jedinki. Što je postignuto rekuzivnim proveravanjem „rupa" u prostoru. Ukoliko se oko cilja nalazi velika grupa jedinki, jedinka sa spoljnih slojeva grupe pokušava da se približi cilju. Ona rekurzivno proverava sve jedinke u toj grupi sve dok neka od njih ne „javi" da može negde da se pomeri. To jest da je nađena „rupa" u grupi entiteta. Tada se entitet približi cilju a „rupa" se popuni. Ovaj proces traje sve do stabilizacije svih jedinki, do optimalnog popunjavanja prostora i „smirivanja" svih entiteta.

Primer:

Svi entiteti (slika 5) se kreću ka cilju (X, slika 5).Provera počinje od primarnog entiteta (slovo P u sredini, slika 5). Proverom je utvrđeno da je početni entitet u sudaru sa dva entiteta. Ta dva entiteta postaju njegova „deca" (entiteti sa jedinicom u sredini, slika 5). Pošto početan smer kretanja dece ulazi u putanju svog roditelja to jest primarnog entiteta, mora da se promeni brzina kretanja. Prvo se testira dete levo od početnog entiteta. Testiraju se brzine normalne na redovnu brzinu i obe se sudaraju sa entitetima, sa tim što se jedna od normalnih brzina sudara sa svojim roditeljem i ona se odbacuje. Druga normalna brzina se sudara sa drugim entitetom (entitet broj 2, slika 5) i taj drugi entitet se označava kao dete. Pošto njegova putanja dovodi do sudara njega i njegovog roditelja, proverava se validnost jedne od brzina normalnih na originalnu, kada se ustanovi da nema sudara sa leve strane deteta (entitet br. 2). Dete svom roditelju (levi entitet br. 1) vraća odobrenje za ažuriranje, a njegov roditelj



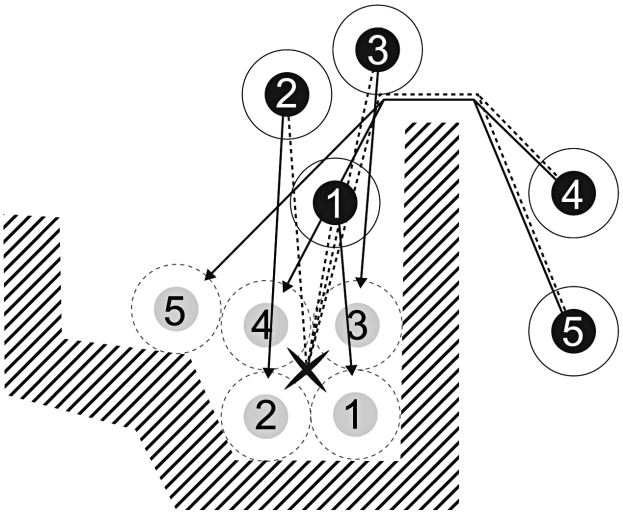
Slika 5.
Drugi algoritam izbegavanja sudara izmedu entiteta

Figure 5.
The second algorithm for collision avoidance between entities

(levi entitet br. 1) potom vraća odobrenje svom roditelju, to jest primarnom entitetu. Međutim početni primarni entitet ima dva deteta, i do sad je jedno vratilo dozvolu. Da bi entitet dobio dozvolu za ažuriranje, sva njegova deca moraju da vrate dozvolu. Tada se proverava drugo dete (desni entitet br. 1), i kada ono vrati dozvolu tada entitet može da se ažurira.

5. Grupisanje entiteta oko cilja

Ne mogu svi entiteti da dođu do cilja, to može samo jedan, onaj koji prvi stigne. Nakon toga se ostali entiteti grupišu oko njega. Ovo grupisanje može da se izvede na različite načiine, i da predstavlja problem pri detekciji ciljnog stanja. Rešenje ovog problema bi bilo automatsko planiranje pozicije za svaki entitet. Na početku simulacije se oko cilja unapred određuju pozicije do kojih entiteti moraju da dođu, jer na poziciji cilja može stati samo jedan entitet. Pozicije su numerisane, i kada se bilo koji entitet približava cilju dodeljen mu je broj pozicije na koju treba da stane. U suprotnom bi entiteti zauzimali pozicije koje su im najbliže. Time bi neke pozicije ostale neiskorišćene a neki entiteti ne bi imali gde da stanu („loše pakovanje"). Na primer na slici 6, pet entiteta se kreće ka cilju. Pozicije su unapred izračunate (sivi krugovi sa isprekidanim ljuskama). Entiteti su numerisani u odnosu na dužinu svoje putanje do cilja (isprekidane linije). Entitet sa najkraćom putanjom do cilja se kreće ka poziciji br. 1. Kreće se novom putanjom (puna linija sa strelicom) a entitet posle njega ka poziciji br. 2, itd. Ovaj način grupisanja je idealan ali ne liči ni na jedan prirodni fenomen. Iz tog razloga se on ne koristi u ovom projektu. U ovom projektu u svrhu grupisanja služi drugi algoritam izbegavanja sudara.

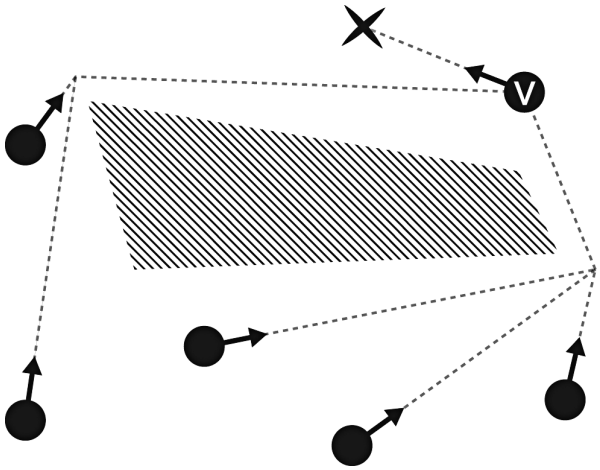


Slika 6.
Grupisanje oko cilja, automatsko planiranje pozicije za svaki entitet

Figure 6.
Goal grouping, automatic position planning for each entity

Algoritmi grupnog kretanja

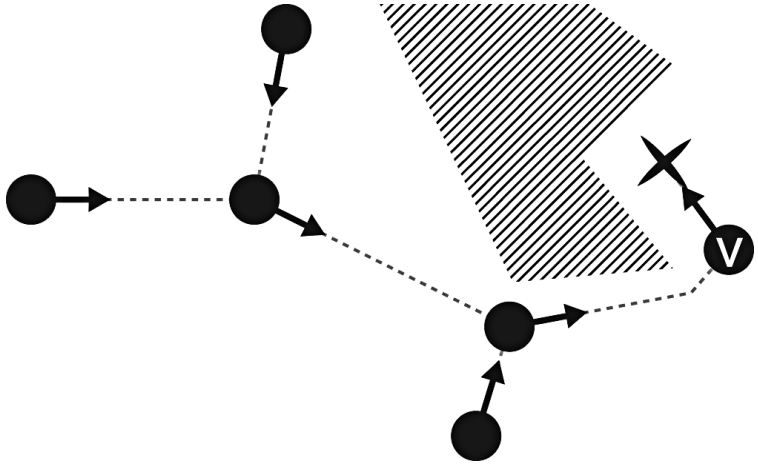
1. Praćenje vođe. Ovaj algoritam određuje objekat koji je najbliži cilju i njega smatra za vođu (entitet sa slovom V, slika 7). Vođa računa svoju putanju i kreće se samostalno. Kada vođa počne da se kreće svi ostali objekti računaju najbližu putanju do vođe u datom trenutku. Da bi njihovo kretanje bilo što efikasnije, potrebno je što češće računati putanju do vođe. Bilo bi idealno da se ove putanje računaju u svakom koraku iteracije. Međutim, to u ovom slučaju nije izvodivo, jer bi računanje više A* algoritama svake iteracije znatno usporilo izvršavanje aplikacije, pogotovo ako postoji veći broj objekata (slika 7).



Slika 7.
Praćenje vođe

Figure 7.
Following the leader

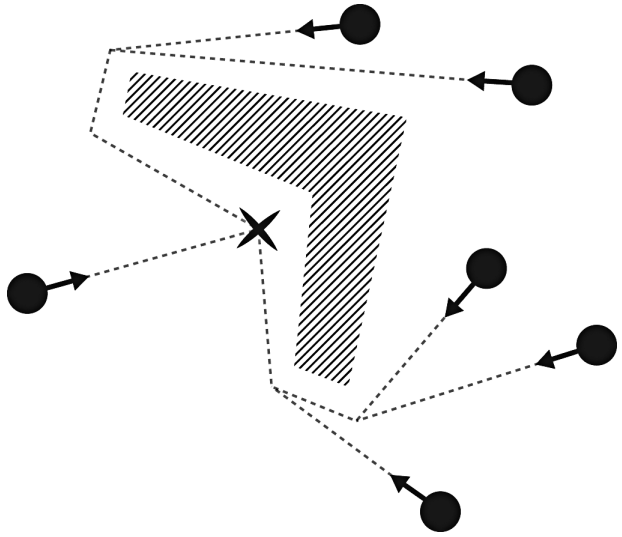
2. Praćenje najbližeg do sebe. Jedina razlika između ovog i prethodno opisanog algoritma je ta da u ovom algoritmu svaka jedinka ne računa najkraći put do vođe, već računa put do objekta koji je njoj najbliži. U ovom algoritmu takođe postoji vođa grupe (entitet obeležen slovom V, slika 8). On je entitet koji zna put do cilja. Entitet koji je njemu najblizi prati njega, i tako redom. Tako se pokreću svi entiteti i svi se kreću ka cilju lančano prateći vođu. Pri odabiru najbližeg objekta dolaze u obzir samo oni koji su bliži lideru nego entitet koji ih bira. Još jedna stvar koju ovaj algoritam ne uzima u obzir je vidno polje jedinke. Entitet traži sebi bliskog entiteta, ne gleda samo entitete ispred sebe. Vidno polje entiteta nema smisla implementirati u ovom projektu jer su entiteti predstavljeni materijalnim tačkama koje nemaju orijentaciju. Samim tim ne mogu ni imati vidno polje, koje zavisi od orijentacije – to jest, vidno polje je 360°. Što se računanja putanje do najbližeg entiteta tiče, važi isto kao i za prvi algoritam: što češće to bolje, idealno bi bilo u svakom koraku iteracije, ali to bi bilo veoma računski zahtevno, naročito ako bi imali veći broj entiteta (slika 8).



Slika 8.
Praćenje najbližeg do sebe

Figure 8.
Following the closest entity

3. Svaka jedinka zna put do cilja. Ovo prakitčno nije algoritam grupnog kretanja. Međutim, biće koristan jer će moći da se upoređuje u odnosu na prethodna dva. U ovom algoritmu svaki entitet nezavisno računa put do cilja. Dobra strana ovog algoritma je što se putanja računa samo jednom na početku, za razliku od ostala dva algoritma (slika 9).



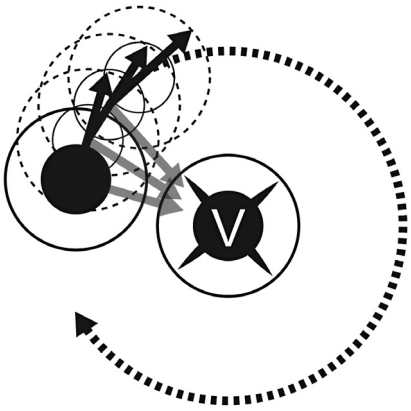
Slika 9.
Svaki entitet zna put do cilja

Figure 9.
Every entity knows the path to the goal

Nalaženje kriterijuma za izlaz algoritma

Kriterijum za izlaz algoritma je trenutak kada su svi entiteti maksimalno popunili prostor oko cilja i kada je kretanje svedeno na minimum. Drugim rečima, kada su se entiteti stabilizovali. Prvobitna ideja je bila da kriterijum za izlaz bude kada svi entiteti vrate zabranu ažuriranja. Problem predstavljaju dve pojave izazvane činjenicom da kada svi entiteti dođu do cilja, neki i dalje imaju smerove u kojima mogu da se kreću:

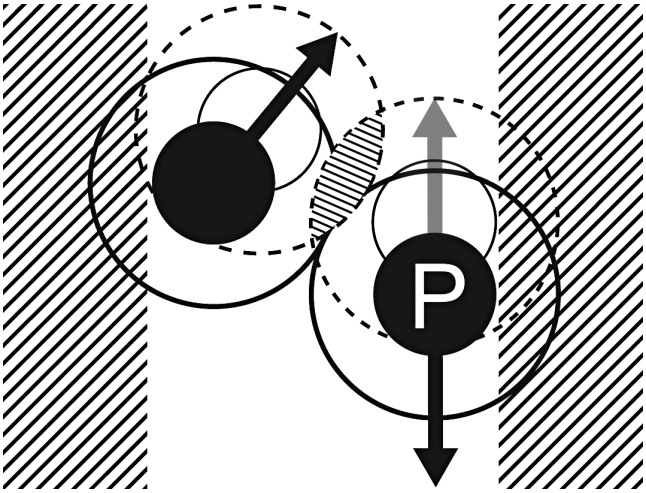
1. Kružno kretanje. Vođa grupe (označen slobom V) (slika 10) kada stane na cilj (X) prestaje sa kretanjem. Svaki entitet koji dođe posle njega takođe „želi" da stane na njegovo mesto. Na početku svake iteracije se računa željenja brzina kretanja (sivi vektor brzine prema cilju, slika 10). Potom se algoritmom izbegavanja sudara ta brzina okrene za 90°. Tako se dobija kružno kretanje oko cilja „po tangenti" i to nije slučaj samo sa jednim entitetom. I kada je mnogo više entiteta oko cilja, dešava se njihovo kružno kretanje ukoliko nema nekih prepreka koje bi mogle to da spreče. Kretanje je kružno zato što se pamti prošli smer kretanja, to jest, ukoliko je entitet jednom odabrao jedan od dva alternativna normalna smera u odnosu na brzinu kretanja, taj smer će biti odabiran dok god je dostupan.



Slika 10.
Kružno kretanje

Figure 10.
Circular motion

2. Kretanje „u nazad". Pored kružnog kretanja takođe se dešava kretanje unazad. Entitet teži da dođe do cilja, ali ne može, a ne može ni jednim ni drugim alternativnim smerom. Tada se kreće unazad, mada samo privremeno, jer se odmah u sledećoj iteraciji ponovo izračunava vektor brzine ka cilju. Iako privremeno, ovo kretanje unazad uzrokuje da svi entiteti zajedno nikada neće vratiti zabranu ažuriranja. Neki entiteti hoće ako su u sredini grupe entiteta, ali će uvek biti onih entiteta koji su na ivicama grupe. Oni će uvek ići unazad, pa ponovo napred, oscilujući. Prvobitni algoritam izbegavanja sudara nije dozvoljavao kretanje unazad. To je izmenjeno jer je primećen slučaj (slika 11) u kome bi se entiteti „zaglavili" da ne postoji opcija kretanja unazad. Ulaz algoritma je primarni entitet (slovo P, slika 11), smer kretanja primarnog entiteta je sivi vektor, smer kretanja drugog entiteta je crni vektor (slika 11). Prvi algoritam izbegavanja sudara ne vraća novu brzinu. Svi smerovi su zauzeti (smer kretanja je u sudaru sa drugim entitetom, alternativni smerovi su u sudaru sa preprekom i sa drugim entitetom). Pošto prvi algoritam nije uspeo, primarni entitet se prosleđuje drugom algoritmu. Drugi algoritam



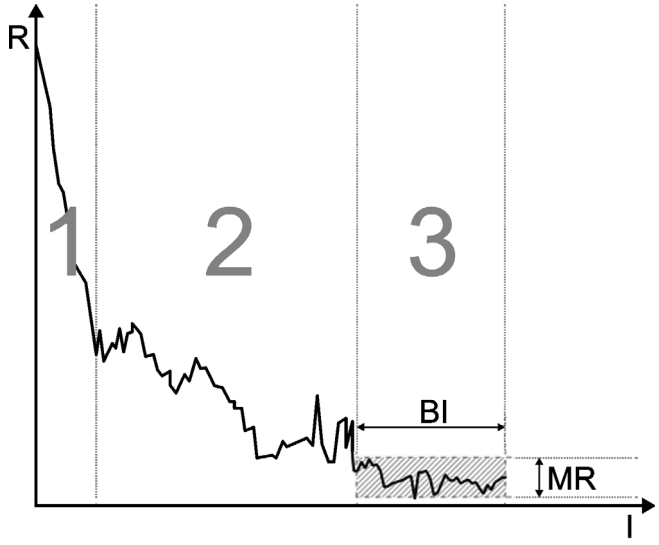
Slika 11.
Kretanje unazad

Figure 11.
Going backwards

na osnovu buduće pozicije primarnog entiteta uzima drugi entitet kao svoje dete. Drugi entitet nema gde da se pomeri a da ne dođe u sudar ili sa preprekom ili sa svojim roditeljem tako da vraća zabranu ažuriranja. Tada drugi algoritam proverava alternativne brzine (obe za po 90° u odnosu na brzinu kretanja primarnog). Jedna je u sudaru sa preprekom, pa je odmah odbačena, a druga kao dete uzima isti entitet kao i brzina kretanja. Pošto je dete isto, i rezultat je isti. Ponovo je vraćena zabrana azuriranja. Jedini izlaz iz ovakve situacije je poslednji korak drugog algoritma. To je provera brzine suprotne brzini kretanja, to jest „kretanje u nazad" (crni vektor primarnog entiteta), što je jedini izlaz iz ove situacije. Primarni entitet dobija novu brzinu i ažurira se njom „oslobađajući" put drugom entitetu koji nastavlja svojim putem. U ovom slučaju je drugi entitet dobio prednost, a primarni entitet se u jednoj iteraciji pomerio u nazad da bi već u sledećoj iteraciji nastavio redovnom putanjom.

Prosečna razdaljina od cilja

Ova metoda rešava dve gore navedene pojave, to jest uspeva da uprkos njima nađe stabilno stanje, to jest kriterijum za izlaz algoritma. Glavna ideja ovog metoda je pronalaženje „stabilnog" stanja grupe entiteta. To jest trenutka kada su promene u rasporedu entiteta oko cilja što manje. Tokom svake iteracije (x osa grafika oznacena slovom I, slika 12) „pamti" se prosečno rastojanje svih entiteta od cilja, R. R je zbir rastojanja svih entiteta podeljen sa brojem entiteta (y osa na grafiku, slika 12). Na slici 12 prikazan je grafik zavisnosti prosečnog rastojanja do cilja i proteklog vremena to jest iteracija. Algoritam koji proverava stabilnost sistema uzima dva parametra: maksimalna razlika rastojanja MR, i vremenski interval (broj iteracija) BI. Grafik je podeljen na tri zone (1, 2, 3). U prvoj zoni se vidi drastičan pad R-a. Ovo je zato što R počinje



Slika 12.
Grafik prosečnog rastojanja entiteta do cilja u zavisnosti od vremena (iteracija)

Figure 12.
Graph of the average distance of all entities to the goal through time (iterations)

da se beleži kada prvi entitet dođe do cilja. Tada je R prilično velik, ali opada jer kada prvi entitet dođe do cilja ostali entiteti su iza njega i brzo napreduju jer nisu jos počeli mnogo da se sudaraju. Prva zona prelazi u drugu zonu kada svi entiteti počnu međusobno da se sudaraju i raspoređuju, to jest kada su svi blizu cilja. U drugoj zoni se entiteti dejstvom algoritma izbegavanja sudara optimalno raspoređuju da bi popunili prostor oko cilja. Tu je pad R-a slabiji ali i dalje prisutan. Treća zona je zona stabilizacije, to je zona gde se R stabilizuje. Da bi algoritam odredio izlaz, to jest stabilno stanje, mora se uzeti u obzir poslednjih BI iteracija. Od poslednjih BI iteracija treba naći najmanju vrednost R-a i najveću vrednost R-a. Ukoliko je razlika najveće i najmanje vrednosti R manja od MR, onda je pronađen izlaz algoritma i entiteti su stabilizovani. U trećoj zoni je šrafiranom površinom ilustrovan deo grafika pomoću kojeg je pronađen izlaz algoritma.

Merenje razdaljine i vremena do cilja

Vreme se meri u iteracijama a razdaljina u pikselima. Svakih N iteracija se na pređenu razdaljinu pojedinačnog entiteta dodaje razlika pozicija nakon ovog vremena. Ukoliko bi se ovo činilo tokom svake iteracije, svaki entitet bi imao približno istu pređenu razdaljinu jer je brzina konstantna a jedini slučaj kada se entitet ne ažurira je kada nije u mogućnosti nigde da se pomeri bilo zbog prepreka ili drugih jedinki. Vreme iskazano u iteracijama se dodaje takođe svakih N iteracija, jer bi i vreme bilo prilično ujednačeno kada bi se dodavalo nakon svake iteracije. Posle N iteracija se proverava promena u poziciji. Ukoliko je veća od neke određene vrednosti koja se smatra dovoljnim pomerajem, onda se

dodaje vreme. Ukoliko nije onda se smatra da je entitet samo „pulsirao", to jest da se kretao sve vreme ali čas na jednu čas na drugu stranu, to jest oscilovao oko nekog položaja. U tom slučaju on nije prešao nikakvu korisnu razdaljinu i verovatno se nalazi u grupi entiteta oko cilja, što je krajnja destinacija njegovog kretanja.

Zaključak

Na osnovu velikog broja testova i raznih konfiguracija terena. Ustanovljeno je da je treći algoritam pokazao najveću efikasnost u pogledu pređene razdaljine do cilja i proteklog vremena do stabilizacije, što je bilo i očekivano. Posle njega najbolje se pokazao prvi a najlošije drugi.

Na osnovu ovih rezultata zaključak je da efikasnost algoritma raste sa količinom „znanja" koje pojedinačni entitet ima. U trećem algoritmu entiteti znaju put do cilja, to je najveća količina znanja. Posle trećeg najviše znanja imaju entiteti u prvom algoritmu. U prvom algoritmu entiteti znaju put do vođe, to jest znaju put do entiteta koji zna put do cilja. A u drugom je količina znanja najmanja, stoga je i efikasnost najmanja. U drugom algoritmu entiteti ne znaju ni put do cilja ni put do vođe, nego prate one entitete koji su im najbliži.

Literatura

web 1: http://www.policyalmanac.org/games/aStarTutorial.htm

web 2: http://www.codezealot.org/archives/55

Vladimir Makarić

Comparison of the Efficiency of Simulations of Group Movement

Group movement is defined as the movement of a group of entities such that all the entities simultaneously move towards a common goal. An entity is an abstract term denoting any of the various participants in group movement, e.g. a bird in a flock, an antelope in a herd, a tank on a battlefield etc. The aim of this project is to compare the efficiency of the algorithms that simulate group movement. The entities are represented in the simulation as material points with a fixed, non-zero minimal distance between any two entities. The entities move in a two-dimensional space, between start and goal points, and obstacles may be present. The first al-

gorithm is based on having all the entities follow one designated leader entity, the second algorithm has each entity follow its most proximal one, and the third algorithm has each entity moving individually, without regarding other entities. The criteria used to compare these algorithms were cumulative distance travelled and total duration of movement of all the entities. In an efficient algorithm both of these measures are as small as possible. The third algorithm was expected to be the most efficient according to these criteria. Testing showed that the most efficient algorithm was the third one, then the first one, and the worst results were produced by the second algorithm.