SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2405

VIZUALIZACIJA SIMULACIJE KRETANJA MNOŠTVA LJUDI

Tin Srnić

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2405

VIZUALIZACIJA SIMULACIJE KRETANJA MNOŠTVA LJUDI

Tin Srnić

Zagreb, lipanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

DIPLOMSKI ZADATAK br. 2405

Pristupnik:	**Tin Srnić (0036491730)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Vizualizacija simulacije kretanja mnoštva ljudi**

Opis zadatka:

Proučiti simulacijski model potreban za simulaciju kretanja mnoštva objekata. Posebice obratiti pažnju na moguće sudare te mimoilaženje pojedinih objekata. Definirati različita ponašanja u kontekstu kretanja. Implementirati simulaciju i vizualizaciju kretanja mnoštva ljudi i zadanoj sceni. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti programski jezik C++, API OpenGL te GLFW i GLEW knjižnice. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 28. lipnja 2021.

Sadržaj

Uvod

Ovaj rad bavi se nekolicinom algoritama inteligencije roja (engl. Swarm intelligence) s naglaskom na kretanje gomila ljudi. Takvi algoritmi fokusiraju se na simuliranje kretanja pomoću velikog broja čestica ili agenta. Brojevi čestica mogu biti u rasponu od par tisuća pa da više milijuna, dok brojevi agenta ovise o primjeni, no u slučaju simulacije mnoštva ljudi odgovaraju količini ljudi koje očekujemo na području simulacije. Sav kod rada može se naći na poveznici [1].

Pojam inteligencije roja uveli su Gerardo Beni i Jing Wang 1989. godine. [2] Njihov rad bavio se staničnim robotskim sustavima te ističe kako takvi sustavi mogu rezultirati netrivijalnim i inteligentnim ponašanjem. U suštini, takvi sustavi temelje se na tome da je u sustavu mnogo agenta koji se kreću ili obrađuju neke podatke na temelju grupe jednostavnih pravila. Promjenom tih pravila ili podešavanjem njihovih parametara, postići će se drugačija konfiguracija agenata.

Ono što inteligenciju roja čini zanimljivom je to što uspješno modelira neke od kompleksnijih sustava koje možemo naći u prirodi. U ovom radu će se prezentirati algoritmi koji simuliraju kretanje riba ili ptica te razvoja pljesni. Takva komplicirana kretanja moguće je simulirati pomoću veoma jednostavnih pravila.

Iako se ne smatra algoritmom inteligencije roja, ovaj rad će započeti opisom Igre života (engl. Game of life) koju je razvio John Conway 1970. godine. Igra života je primjer staničnog automata (engl. Cellular automata) i dijeli mnoge sličnosti s ostatkom algoritama u ovom radu.

Uz Igru života, bit će govora o algoritmu Boida i simulacijama pomoću roja čestica. Algoritam Boida jedan je od prvih algoritama čija je svrha simulacija realističnog kretanja gomila. Iako se njime ne simulira kretanje ljudi, zanimljiv je zbog toga što se jednostavnim pravilima, slično kao u algoritmima simulacije kretanja ljudi, mogu postići složena ponašanja. Najčešće se koristi za prikaz jata ptica ili plova riba. Slično kao i Boidi, simulacije pomoću roja čestica koriste se jednostavnim pravilima. Cilj simulacije nije prikaz kretanja, već iscrtavanje kompleksnih oblika i uzoraka.

1. Igra života

1.1. Općenito o Igri života

Igra života, jednostavnije znana samo kao Život, smještena je u dvodimenzionalnu pravokutnu mreža kvadratnih ćelija. Svaka ćelija može biti živa ili mrtva te budućnost svake ovisi o njenih 8 susjeda. Mreža također postoji u vremenu. Svaki vremenski korak sve ćelije osvježavaju svoje stanje na temelju 4 pravila. [3]

Pravila Igre života:

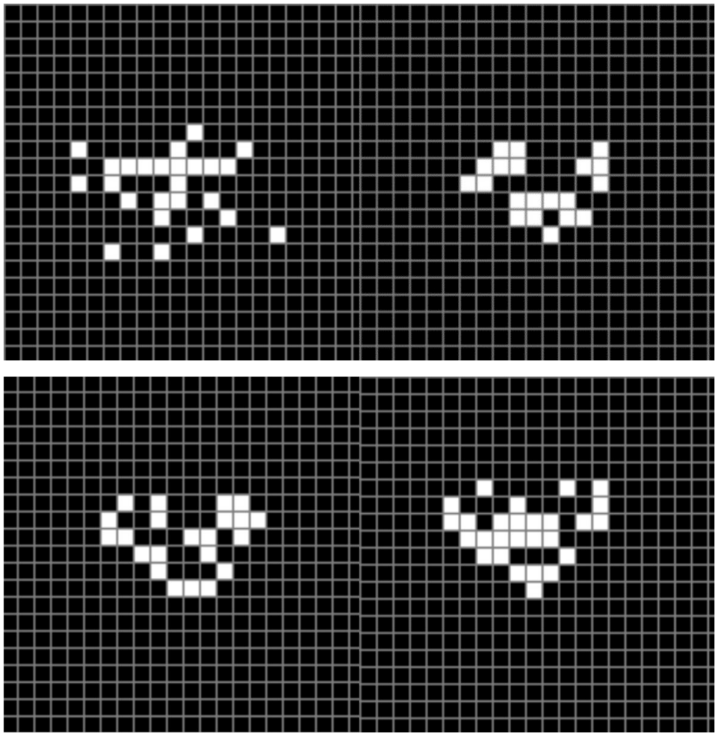
1. Svaka živa ćelija s 2 ili 3 živa susjeda ostaje živa
2. Svaka mrtva ćelija s točno 3 živa susjeda postaje živa
3. Sve ostale žive ćelije postaju mrtve
4. Sve ostale mrtve ćelije ostaju mrtve

Kod u kojem su vidljiva sva 4 pravila prikazan je na slici 1.1.

```
void main() {
    ivec2 id = ivec2(gl_GlobalInvocationID.xy);
    ivec2 coordinates = ivec2(8, 8) * id + ivec2(4, 4);
    int cellStatus = 0;
    if(imageLoad(inTexture, coordinates).x > 0) {
        cellStatus = 1;
    }

    int sum = 0;
    for(int offsetX = -1; offsetX <= 1; offsetX++) {
        for(int offsetY = -1; offsetY <= 1; offsetY++) {
            ivec2 currentCoordinates = ivec2(offsetX * 8, offsetY * 8) + coordinates;
            int cellValue = imageLoad(inTexture, currentCoordinates).x;
            if(!(offsetX == 0 && offsetY == 0) && cellValue  > 0) {
                sum += 1;
            }
        }
    }

    if (sum == 2) {
        colorCell(coordinates, vec4(cellStatus, cellStatus, cellStatus, 1));
    } else if (sum == 3) {
        colorCell(coordinates, vec4(1, 1, 1, 1));
    } else if (sum < 2) {
        colorCell(coordinates, vec4(0, 0, 0, 1));
    } else if (sum > 3) {
        colorCell(coordinates, vec4(0, 0, 0, 1));
    }
}
```

Slika 1.1. Prikaz koda za pravila igre života

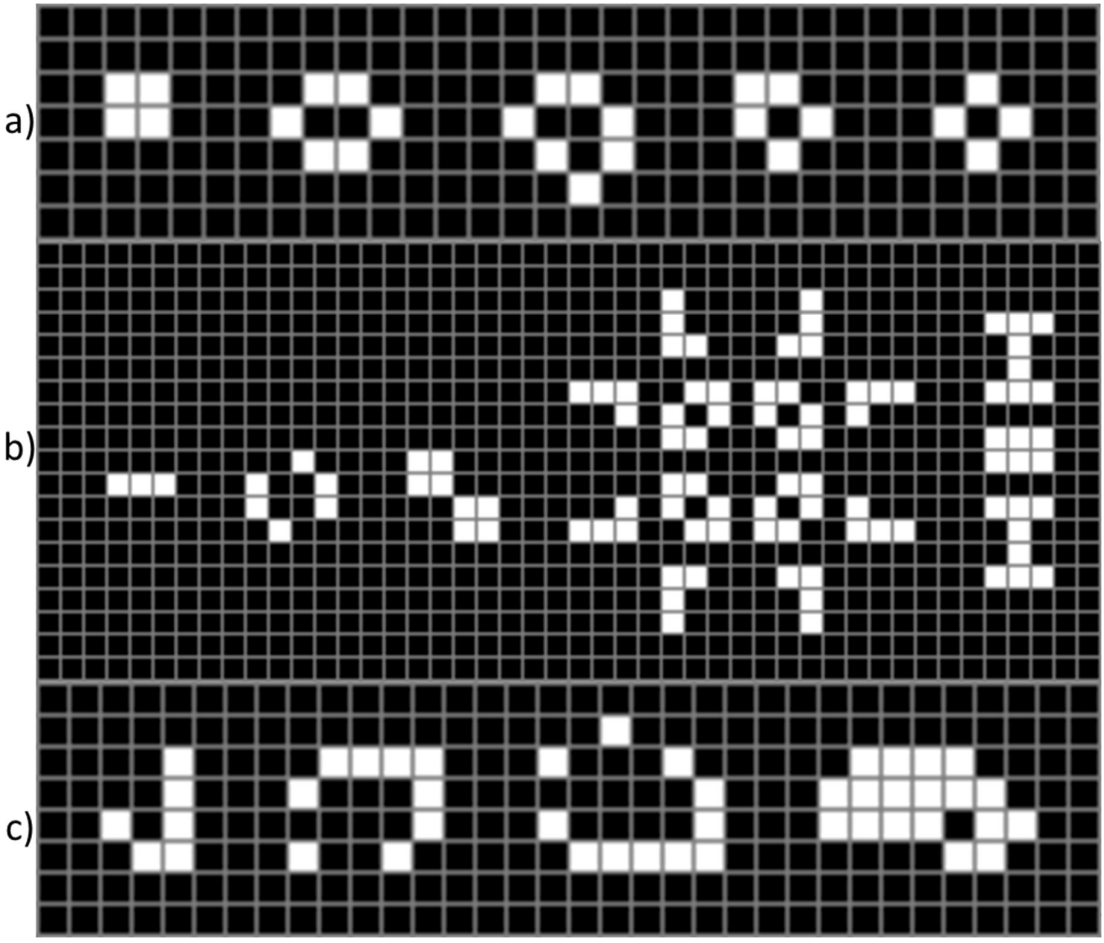
Inicijalni uzorak mrtvih, odnosno živih ćelija, u potpunosti određuje budućnost cijelog sustava, ali iako ova pravila izgledaju jednostavno, omogućavaju veoma kompleksna ponašanja kako je vidljivo na slici 1.2.



Slika 1.2. Prikaz četiri uzastopna koraka u Igri života

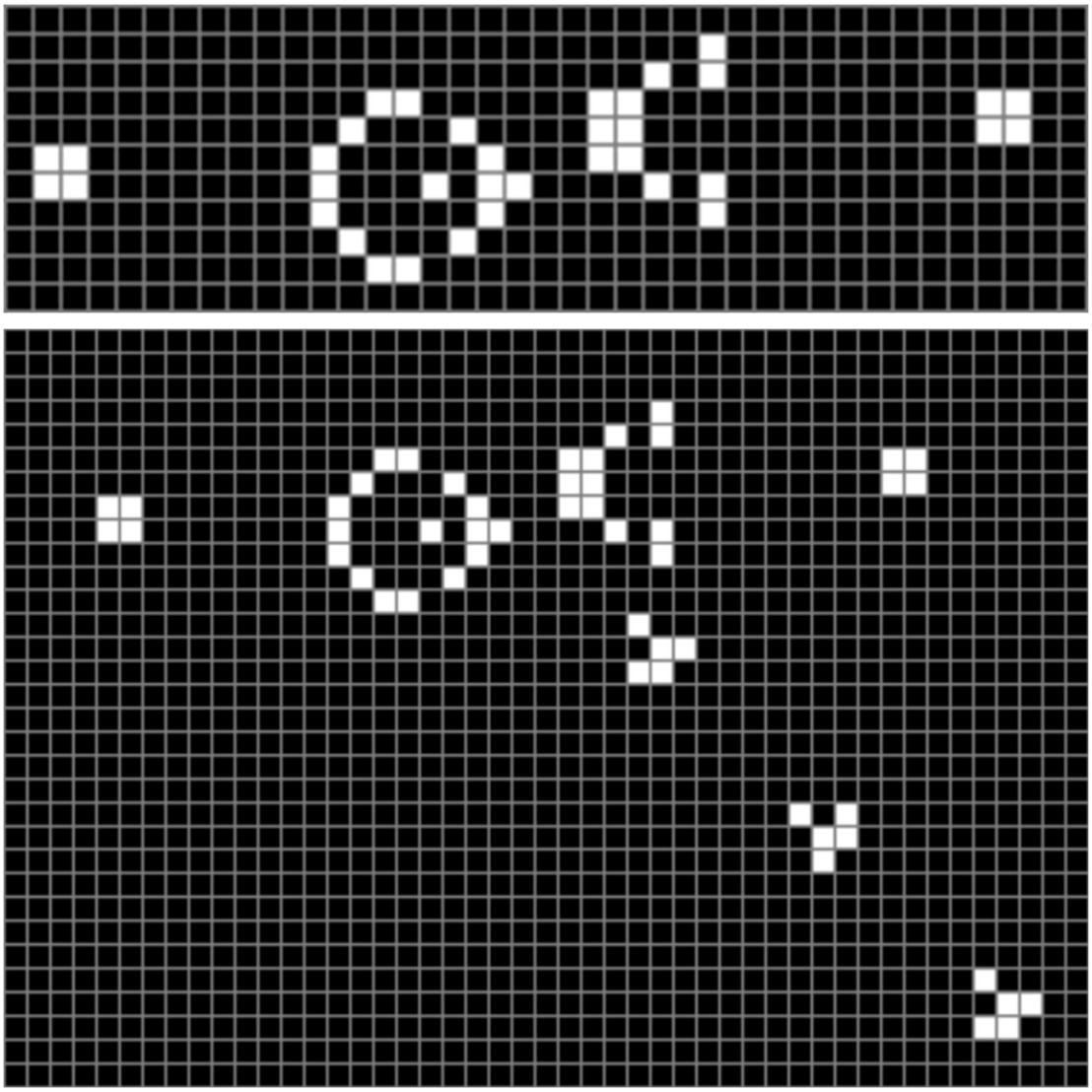
1.2. Karakteristični uzorci

U Igri života javlja se nekoliko posebnih tipova uzoraka koji se klasificiraju prema svome ponašanju. Prvi tip uzoraka su oni koji u svakom koraku ostaju isti, odnosno žive ćelije ostaju žive, a mrtve ostaju mrtve. Sljedeći tip jesu oscilirajući uzorci koji se vraćaju u početno stanje nakon konačnog broja koraka. Postoje i takozvani svemirski brodovi koji se tijekom vremena kreću po mreži. [4] Na slici 1.3 prikazani su neki primjeri navedenih tipova uzoraka.



Slika 1.3. Primjeri tipova uzoraka u Igri života
a) Uzorci koji ostaju isti, b) oscilatori, c) svemirski brodovi

Posebno se ističu uzorci koji se kao i oscilatori periodično gibaju te tijekom vremena stvaraju dodatne uzorke. Primjer ovakvog generatora uzoraka je Gosper jedrilica (engl. Gosper glider gun) čije je početno stanje vidljivo na slici 1.4.



Slika 1.4. Prikaz Gosper jedrilice, a) Početno stanje, b) Stanje nakon 180 koraka

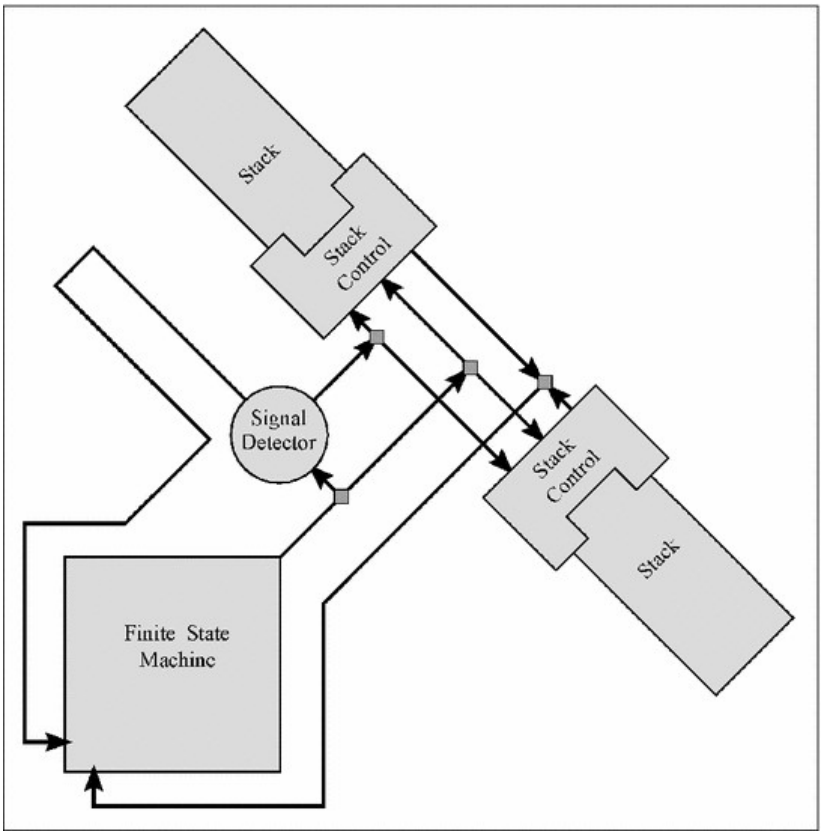
1.3. Neodlučivost igre života

Igra života je neodlučiv algoritam što znači da nije moguće konstruirati algoritam koji bi u konačnom vremenu dao pozitivan odnosno negativan odgovor. U okviru Igre života taj se

problem svodi na pitanje je li moguće, na temelju bilo koja dva stanja igre, odrediti hoće li jedno stanje rezultirati drugim ili obrnuto. [4]

Igra života je ekvivalentna univerzalnom Turingovom stroju (UTM) čiji je model prikazan na slici 1.5. Ukoliko bi postojao algoritam odlučivanja, mogao bi se koristiti za rješavanje problema zaustavljanja (engl. Halting problem). Budući da je poznato da je problem zaustavljanja nerješiv, može se zaključiti da postoje uzorci koji zauvijek ostaju kaotični. Ukoliko to ne bi bio slučaj, svi uzorci mogli bi se pojednostavniti na niz prethodno navedenih karakterističnih uzoraka te bi sva daljnja stanja igre bila poznata.

Simulacija igre života na ploči veličine 80x50 veoma je brza te se postiže 800 do 1100 sličica u sekundi.



Slika 1.5. Shematski prikaz Turingovog stroja

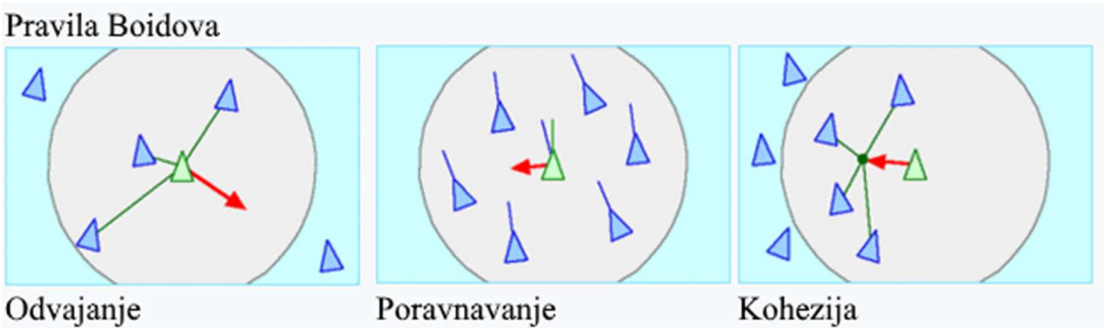
2. Boid algoritam

2.1. Općenito o Boidima

Algoritam Boida razvio je Craig Reynolds 1986. godine. Originalno je zamišljen kao algoritam koji bi simulirao kretanje jata ptica, ali lako se može modificirati za simulaciju kretanja drugih skupina životinja. [5] Algoritam je smješten u dvodimenzionalni prostor popunjen nizom agenata koji, prateći položaj ostalih agenata u blizini, izračunavaju svoju putanju. Brzina kretanja je konstantna, a smjer se određuje pomoću sljedećih pravila:

Pravila kretanja Boidova (slika 2.1.):

1. Odvajanje – svaki agent skreće kako ne bi blokirao put drugom agentu
2. Poravnavanje – agenti se kreću prema prosječnom smjeru kretanja ostalih agenata u blizini
3. Kohezija – agenti se kreću prema centru mase okolnih agenata

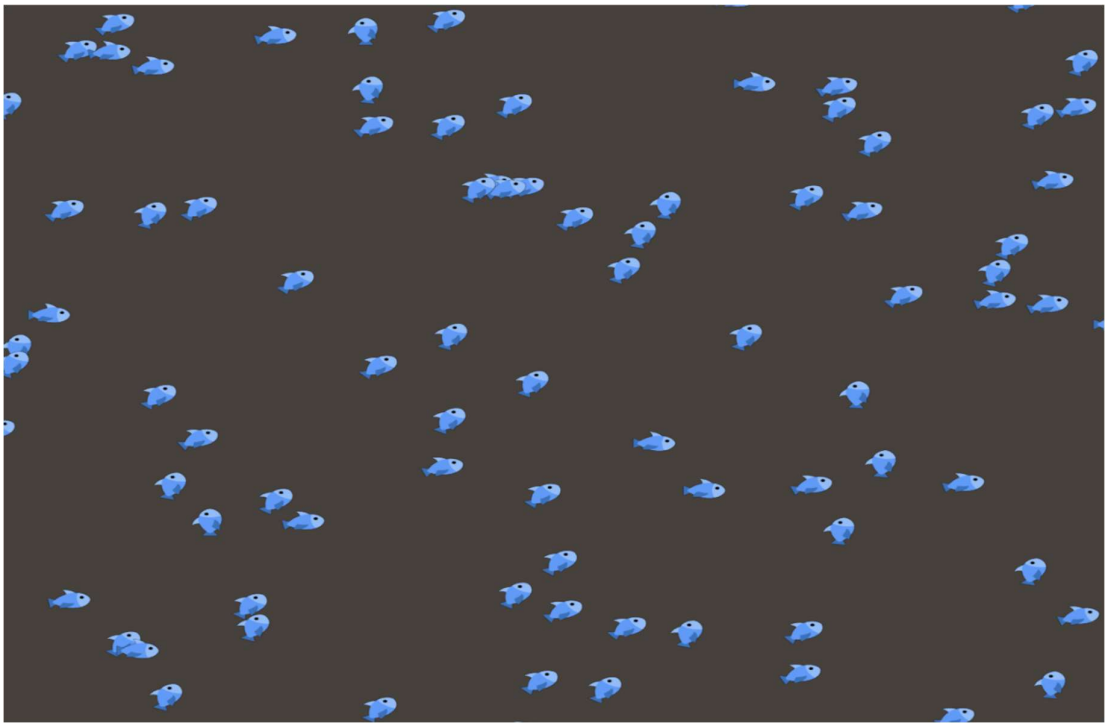


Slika 2.1. Grafički prikaz pravila kretanja agenta

Navedena pravila moguće je nadopuniti dodatnim pravilima poput izbjegavanja prepreka ili traženja cilja kako bi se postiglo realističnije ponašanje. Također, algoritam se može koristiti i za simulacije unutar trodimenzionalnog prostora poput video igara.

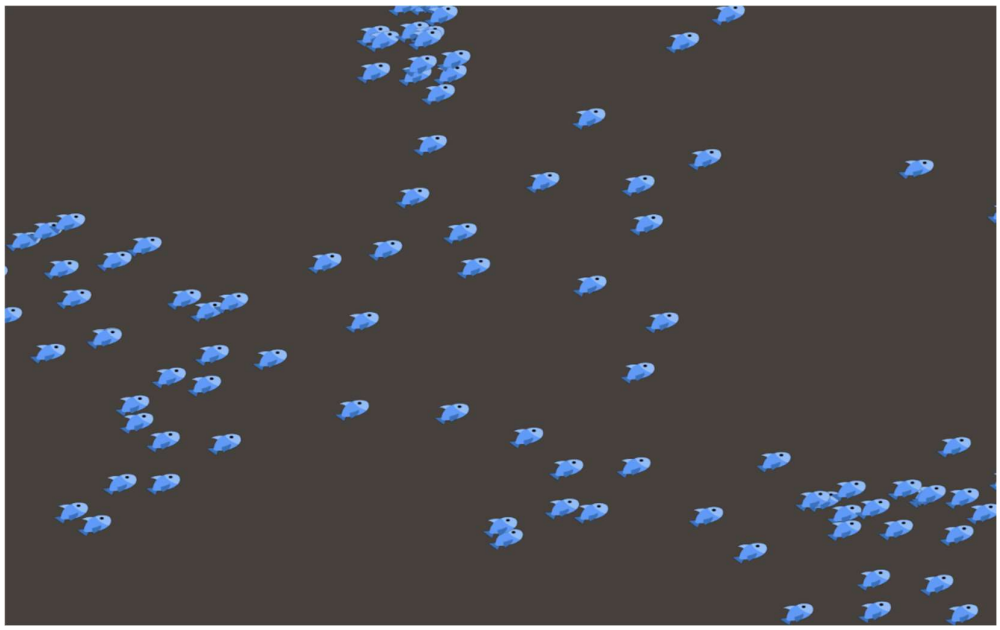
2.2. Utjecaj pojedinih pravila na kretanje Boida

Kako bi se postiglo realistično kretanje Boida, nužna je primjena sva tri gore navedena pravila od jednom. Unatoč tome, korištenjem pojedinih pravila također se postižu zanimljivi uzorci kretanja. Korištenjem pravila odvajanja ne postižu se nikakvi uzorci kretanje te se Boidi samo izbjegavaju. Prikaz kretanja Boidova vidljiv je na slici 2.2.



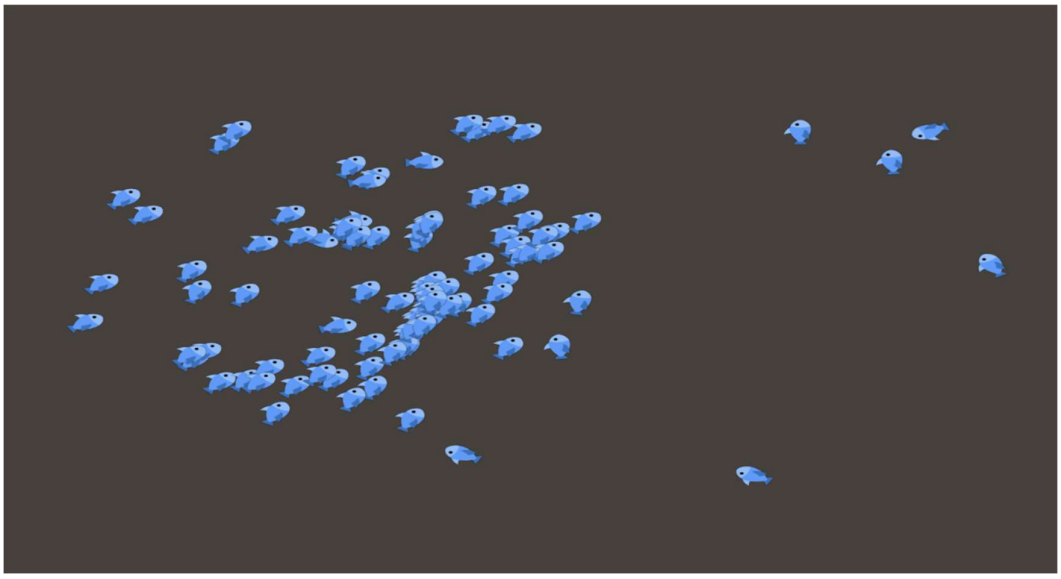
Slika 2.2. Prikaz kretanja Boida uz korištenje prvog pravila

Drugo pravilo drastično mijenja ponašanje Boida. Poravnavanjem se postiže kretanje vidljivo na slici 2.3 koje je znatno sličnije onome koje nalazimo u prirodi. Nedostatak ostalih pravila uzrokuje česte sudare te značajan broj Boida ostaje zasebno.



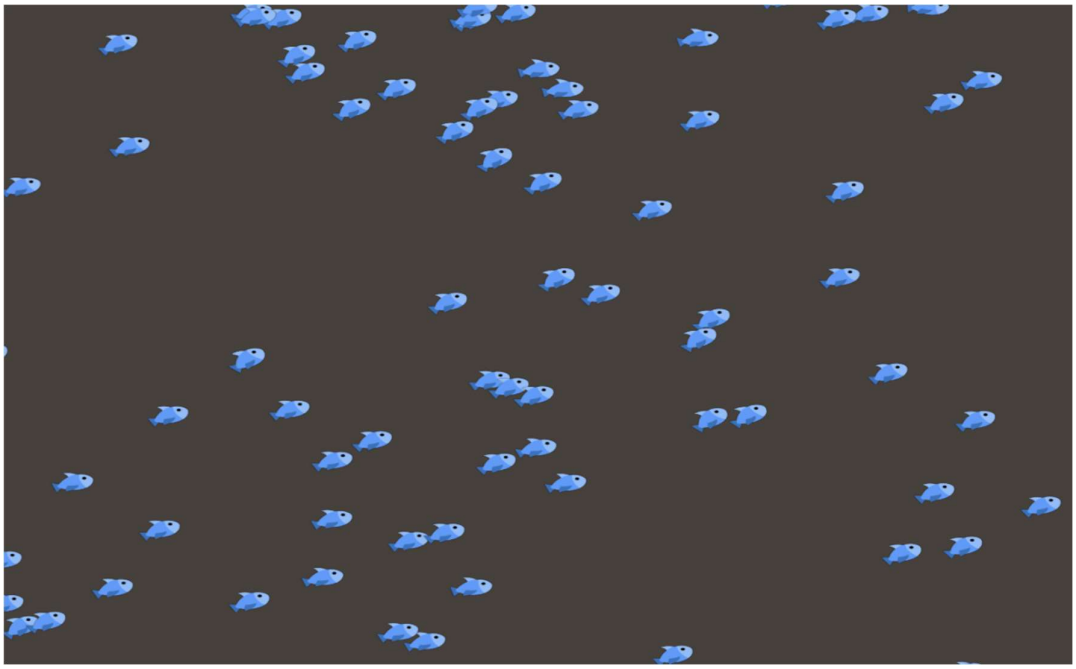
Slika 2.3. Prikaz kretanja Boidova uz korištenje drugog pravila

Primjena pravila kohezije rezultira jako čestim sudarima tijekom kojih se skupina Boida može zaglaviti unutar petlje u kojoj se gibaju spiralno. Također se često javljaju dugi lanci Boida koji prate smjer kretanja Boida s početka lanca. Slika 2.4. prikazuje kod te primjer kretanja Boida s pravilom kohezije.



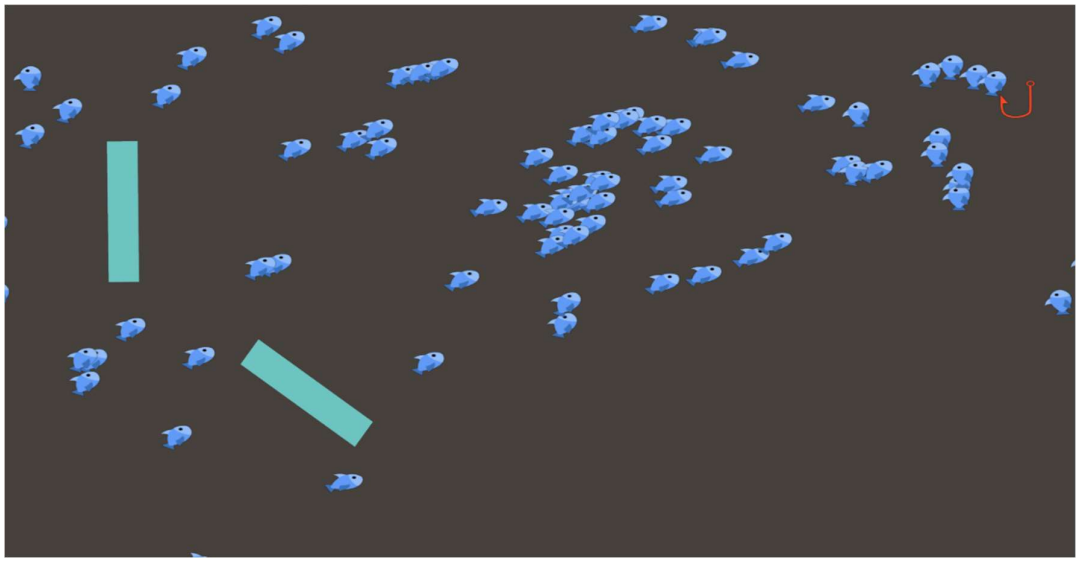
Slika 2.4. Prikaz kretanja Boida uz korištenje trećeg pravila

Iz primjera je vidljivo da je svako pojedino pravilo nedovoljno kako bi se postigao realističan prikaz kretanja. Uz korištenje sva tri pravila, kao što je prikazano na slici 2.5., postiže se realistično kretanje nalik kretanju jata riba.



Slika 2.5. Prikaz kretanja Boida uz primjenu svih pravila

Moguće je dodati prepreke ili cilj kretanja kako je vidljivo na slici 2.6.

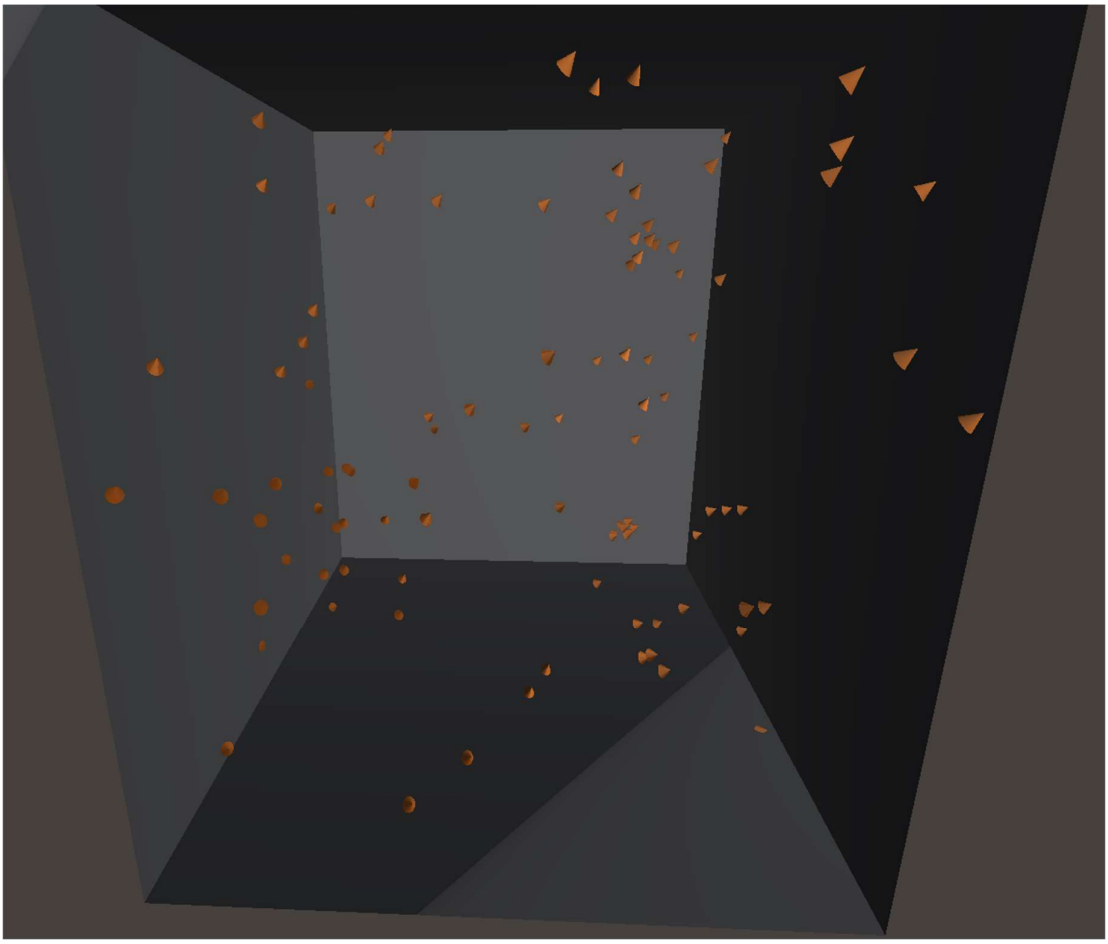


Slika 2.6. Zaobilaženje prepreka (zeleni pravokutnici) i praćenje cilja (crvena udica)

Broj sličica u sekundi za 100 agenata se kreće između 900 i 1000, dok za 1000 agenata pada na 150 do 170, što je očekivano zbog kvadratnog rasta vremenske složenosti.

2.3. Kretanje Boida u trodimenzionalnom prostoru

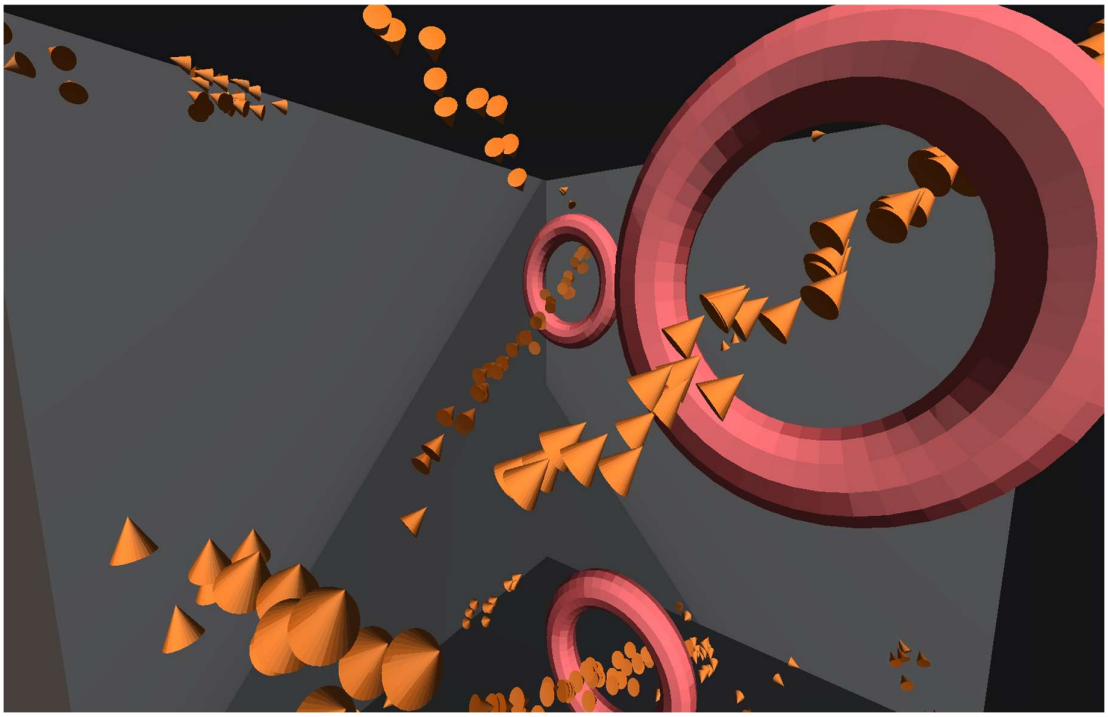
Kako je ranije spomenuto, Boidovi se mogu postaviti i u trodimenzionalni prostor. Pravila njihovog kretanja se bitno ne mijenjaju. Minimalne promijene su potrebne u kodu za pravilo poravnavanja. Zbog postajanja više mogućih osi rotacije u trodimenzionalnom prostoru, prije primjene rotacije na smjer kretanja Boida, mora se odrediti os rotacije. Konačni rezultat kretanja Boida u trodimenzionalnom prostoru vidljiv je na slici 2.7.



Slika 2.6. Boidi u trodimenzionalnom prostru

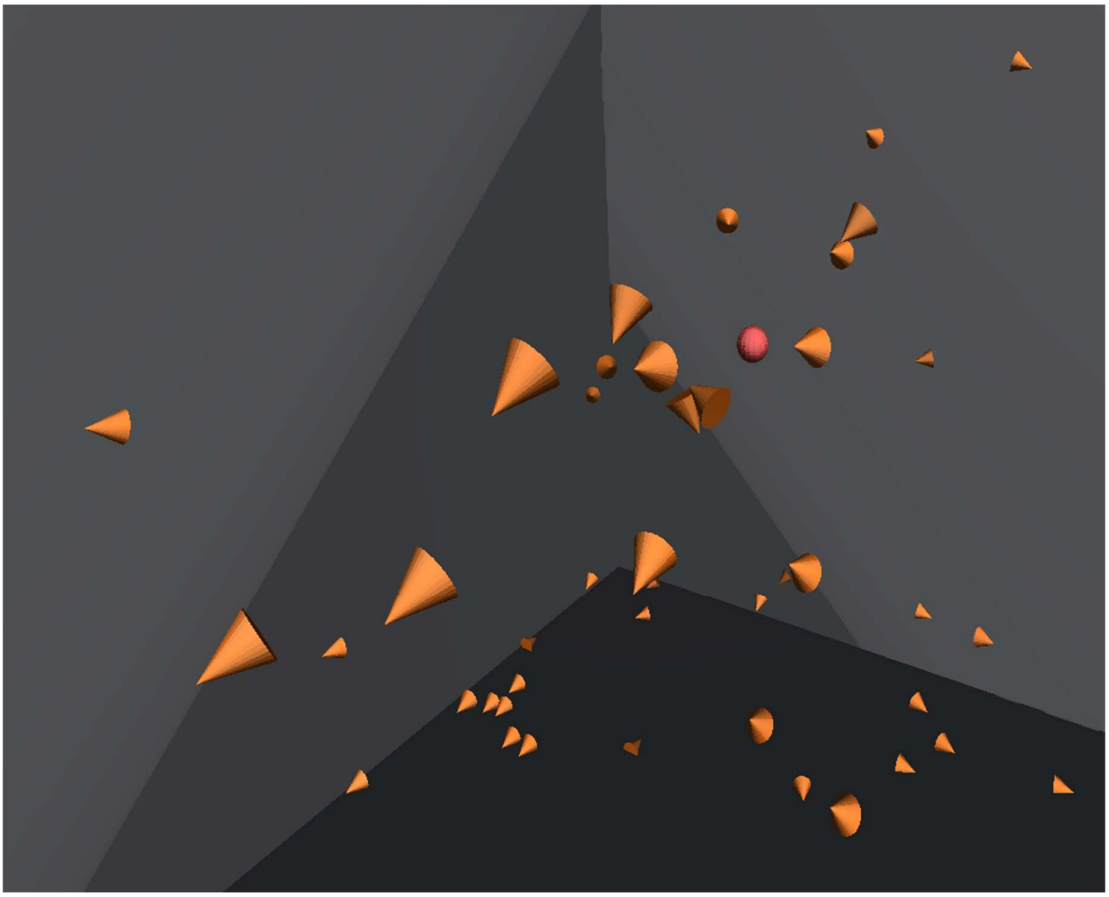
2.4. Dodatna pravila kretanja Boida

U slučaju primjene Boida u kontekstu igara ili simulacija, potrebno je uvesti dodatna pravila kretanja. Kako bi se postiglo realističnije kretanje, Boidi moraju biti sposobni zaobilaziti prepreke na svom putu kao što je vidljivo na slici 2.8.



Slika 2.8. Izbjegavanje prepreka

Također je poželjno uvesti pravilo praćenja mete. Samo pravilo ne utječe na realističnost kretanja Boida, no daje mogućnost kontrole smjera kretanja. Na taj se način u igrama može postići efekt napada ili slijeđenja dominantne jedinke. Primjer takvog kretanja vidljiv je na slici 2.9.

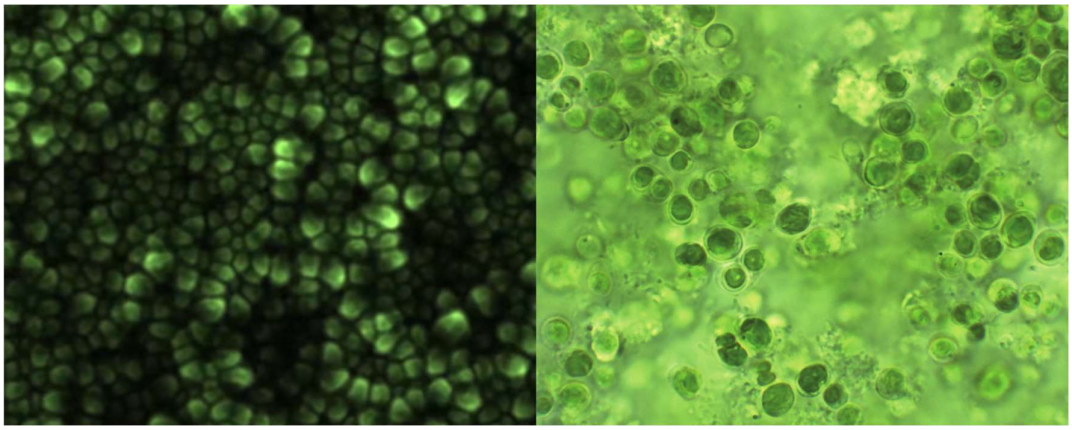


Slika 2.9. Kretanje prema meti (crvena loptica)

3. Simulacije pomoću roja čestica

3.1. Općenito o roju čestica

Nalik Boidima, simulacije pomoću roja čestica koriste jednostavna pravila da bi se postigla složena gibanja. Nasuprot Boidima, koristi se znatno veći broj agenata u rasponu od nekoliko stotina tisuća do više milijuna. Nema praktičnih primjena, ali je zanimljiv za proučavanje zato što se njime mogu postići raznoliki i realistični oblici nalik onima iz prirode kako je prikazano na slici 3.1.

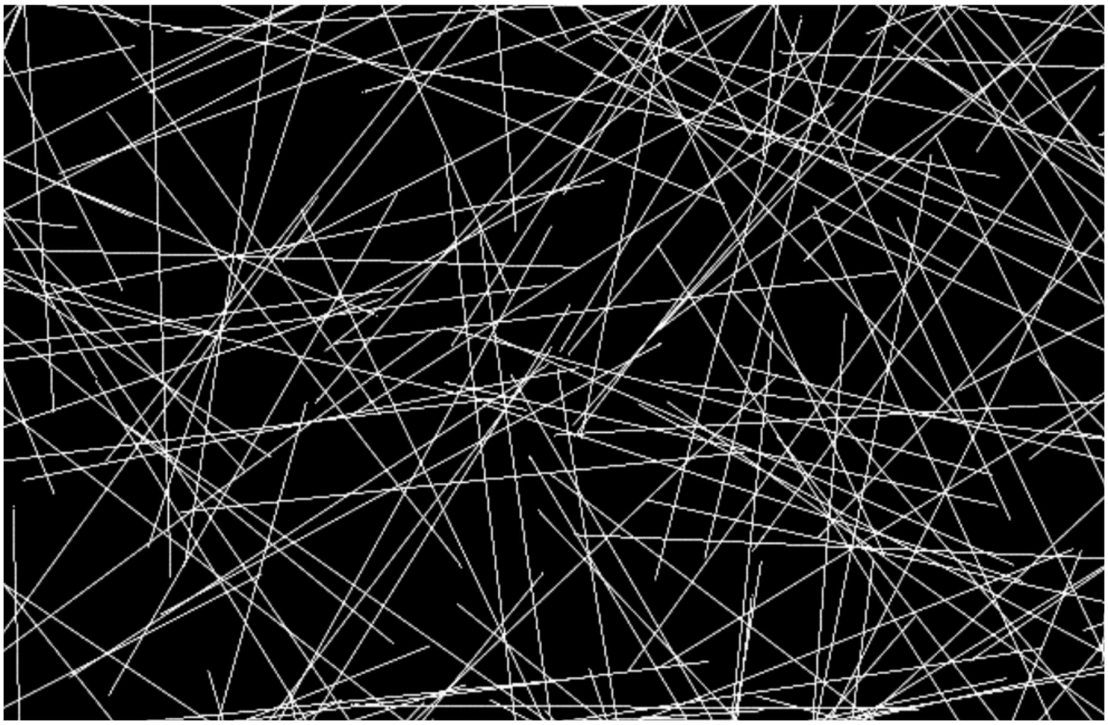


Slika 3.1. Usporedba simulacije (lijevo) i prirode (desno)

3.2. Razvoj kretanja agenta

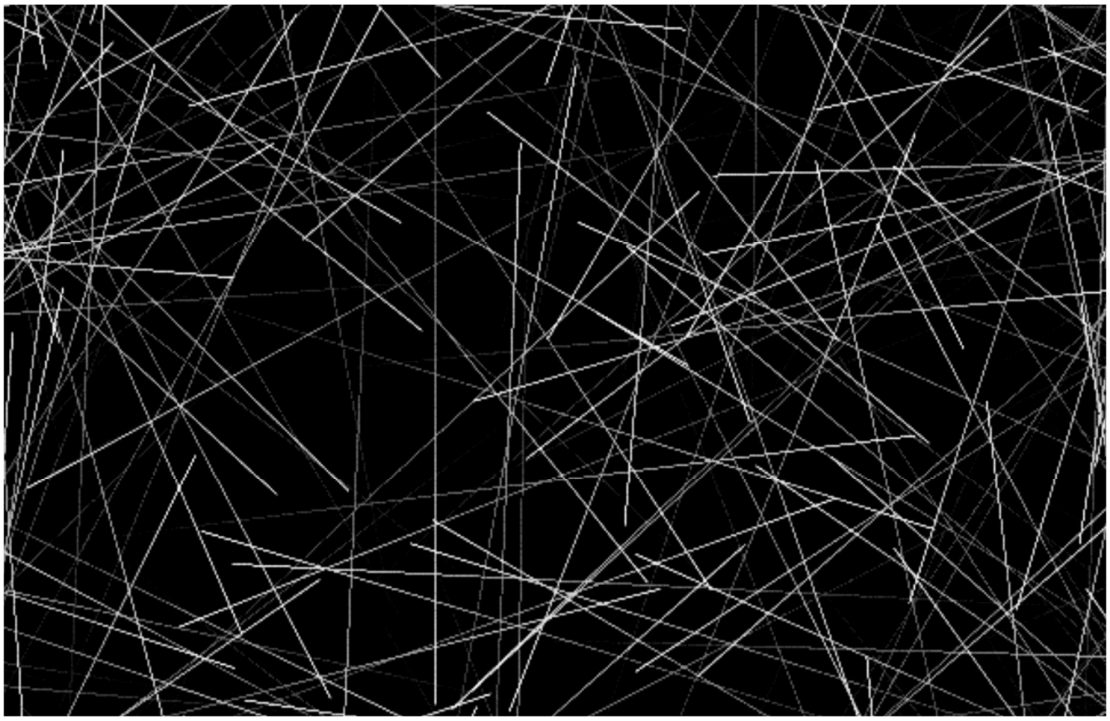
Kako je za simulaciju potrebna velika količina agenta, poželjno je napisati kod za sjenčar za izračunavanje (engl. Compute Shader). Simulacija je veoma pogodna za sjenčar zbog toga što se kod za velik broj agenta može izvoditi istovremeno.

Agenti su smješteni u dvodimenzionalni svijet te svaki ima svoju unaprijed određenu poziciju i smjer kretanja. Kao osnovni oblik gibanja uvodi se pravocrtno gibanje te se pri sudaru s rubom prozora odabire nasumični kut odbijanja. Svaki agent također iza sebe ostavlja trag gibanja kao što je prikazano na slici 3.2.



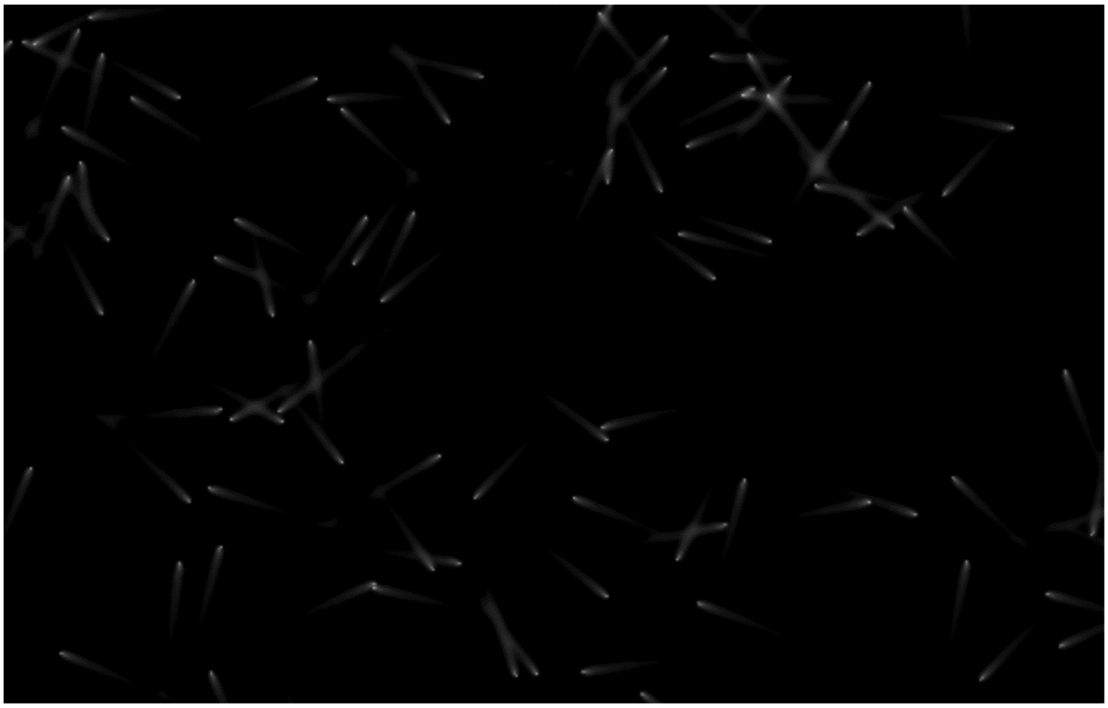
Slika 3.2. Prikaz osnovnog kretanja agenata

U sljedećem koraku uvodi se postepeno nestajanje tragova kako je vidljivo na slici 3.3. Ovo se jednostavno postiže tako da svakom pikselu smanjujemo svjetlinu za neku predodređenu vrijednost. Od velike je važnosti da se taj kod izvodi u sjenčaru zbog velikog broja piksela koji se trebaju obraditi što bi bilo zahtjevno za procesor.



Slika 3.3. Kretanje uz nestajanje tragova

Nadalje je potrebno uvesti raspršivanje traga vidljivo na slici 3.4. Važnost toga je što se u kasnijim koracima agente usmjerava prema tragovima ostalih agenta. Ukoliko bi tragovi ostali veličine jednog piksela, agenti bi ih mogli teže detektirati. Raspršivanje se izvodi tako da se za svaki piksel izračuna prosjek boje susjednih piksela i njega samog. Zbog toga što se u svakom trenutku obrađuje devet piksela, ovaj zadatak je također pogodan za izvođenje na sjenčaru.



Slika 3.4. Kretanje uz raspršivanje traga

Konačno, uvodi se skretanje prema ostalim agentima. Kako bi odredio svoj smjer kretanja, agent promatra tri različite kvadratne skupine piksela. Prva skupina smještena je direktno ispred agenta, dok su ostale dvije lijevo i desno od prve pod proizvoljnim kutom u odnosu na trenutni smjer kretanja agenta. Također je potrebno odabrati udaljenost od trenutnog položaja agenta do centra skupine te veličinu skupine zadanu u broju piksela. Za svaku se skupinu izračunava ukupna količina boje kao što je vidljivo na slici 3.5.

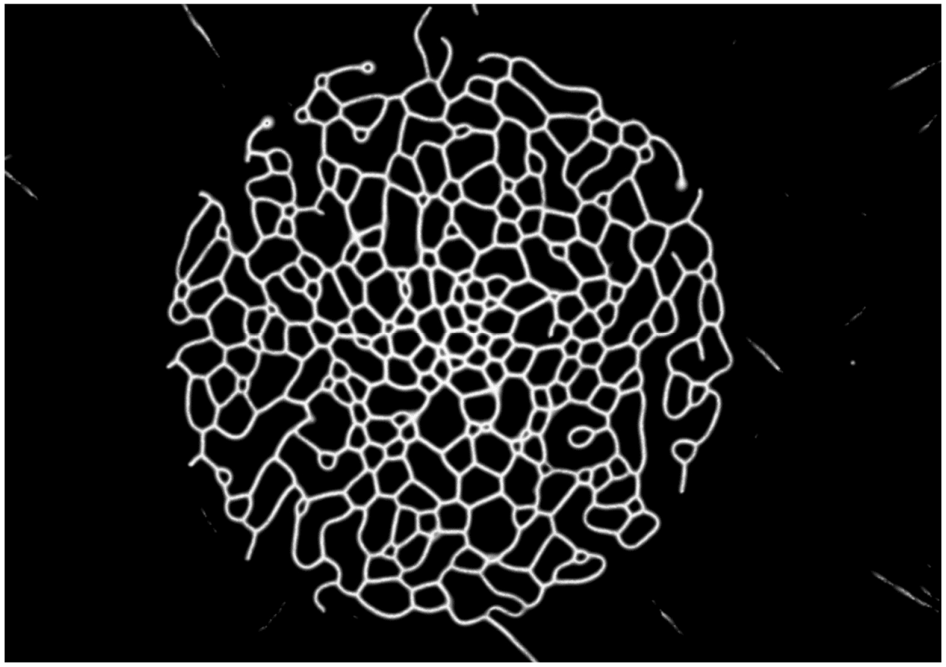
```
float sense(Agent agent, float angle) {
    float sensorAngle = agent.angle + angle;
    vec2 sensorDir = vec2(cos(sensorAngle), sin(sensorAngle));

    vec2 sensorPos = agent.position + sensorDir * sensorOffsetDst;
    ivec2 sensorCentre = ivec2(sensorPos);

    float sum = 0;

    for (int offsetX = -sensorSize; offsetX <= sensorSize; offsetX++) {
        for (int offsetY = -sensorSize; offsetY <= sensorSize; offsetY++) {
            int sampleX = min(width - 1, max(0, sensorCentre.x + offsetX));
            int sampleY = min(height - 1, max(0, sensorCentre.y + offsetY));
            sum += imageLoad(destTex, ivec2(sampleX, sampleY)).x;
        }
    }

    return sum;
}
```

Slika 3.5. Kod za izračunavanje količine boje

Na temelju dobivenih vrijednosti, određuje se smjer kretanja za što je kod prikazan na slici 3.6. Ukoliko je najveća dobiven vrijednost u skupini ispred agenta, njegov smjer se ne mijenja i nastavlja pravocrtno gibanje. Za slučaj da su pojedinačne vrijednosti i lijeve i desne skupine veće od vrijednosti centralne skupine, agent odabire nasumični smjer kretanja koji mora biti unutar 90 stupnjeva sa svake strane od trenutnog smjera. Ako centralna skupina ima vrijednost između onih na stranama, agent skreće prema strani s višom vrijednosti. Kao konačni rezultat gore opisanih koraka, dobiva se kretanje prikazano na slici 3.7.

```
float random = rand(agents[id].position);

float senseAngle = (60.0 / 180.0) * PI;
float weightForward = sense(agents[id], 0);
float weightLeft = sense(agents[id], senseAngle);
float weightRight = sense(agents[id], -senseAngle);
float randomSteer = (rand(vec2(random)) + 1) / 2;

if(weightForward > weightLeft && weightForward > weightRight) {
    agents[id].angle += 0;
} else if(weightForward < weightLeft && weightForward < weightRight) {
    agents[id].angle += (randomSteer - 0.5) * 2 * turnSpeed * deltaTime;
} else if(weightRight > weightLeft) {
    agents[id].angle -= randomSteer * turnSpeed * deltaTime;
} else if(weightLeft > weightRight) {
    agents[id].angle += randomSteer * turnSpeed * deltaTime;
}
```

Slika 3.6. Kod za određivanje smjera kretanja

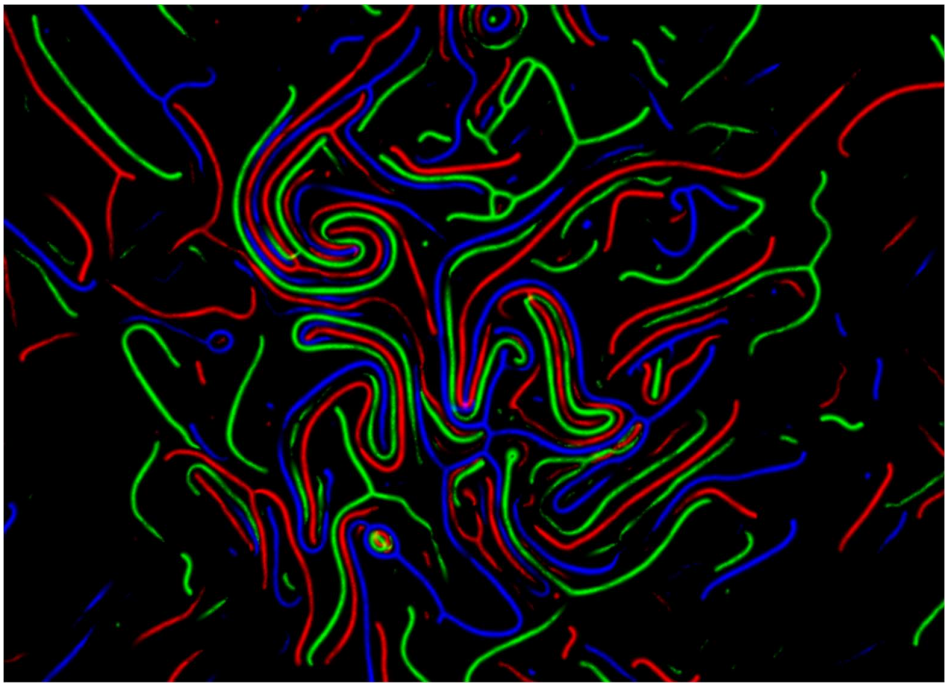


Slika 3.6. Konačni prikaz kretanja

Dodatno je moguće podijeliti agente u podvrste koje su prikazane različitim bojama. Ovom promjenom postiže se dinamičnije ponašanje agenta zbog toga što se različite podvrste natječu. I dalje će se kao do sada određivati kvadratne skupine, no umjesto da se promatra samo ukupna vrijednost boje u njima, boje koje ne odgovaraju podvrsti agenta se smatraju negativnima. To se postiže tako da se kod izmijeni kao što je prikazano na slici 3.7. Konačni prikaz simulacije s više podvrsta agenata vidljiv je na slici 3.8. [6] Za ovaj slučaj postiže se oko 130 sličica u sekundi za 64000 agenta, dok za 640000 agenta samo 15.

```
float sense(Agent agent, float angle) {
    float sensorAngle = agent.angle + angle;
    vec2 sensorDir = vec2(cos(sensorAngle), sin(sensorAngle));

    vec2 sensorPos = agent.position + sensorDir * sensorOffsetDst;
    ivec2 sensorCentre = ivec2(sensorPos);

    float sum = 0;

    vec4 senseWeight = agent.speciesMask * 2 - 1;

    for (int offsetX = -sensorSize; offsetX <= sensorSize; offsetX++) {
        for (int offsetY = -sensorSize; offsetY <= sensorSize; offsetY++) {
            int sampleX = min(width - 1, max(0, sensorCentre.x + offsetX));
            int sampleY = min(height - 1, max(0, sensorCentre.y + offsetY));
            sum += dot(senseWeight, imageLoad(destTex, ivec2(sampleX, sampleY)));
        }
    }

    return sum;
}
```

Slika 3.7. Kod koji uključuje podvrste agenata



Slika 3.8. Prikaz kretanja različitih podvrsta

4. Kretanje mnoštva ljudi

4.1. Općenito o kretanju mnoštva ljudi

Kretanje mnoštva ljudi (engl. Crowd simulation) je način simuliranja kretanja velikog broja jedinki. Najčešće se koristi za stvaranje virtualnih scena u medijima poput filma ili video igara gdje su potrebne velike gomile. Također se koristi u arhitekturi te urbanom planiranju kako bi se osigurao kvalitetan protok ljudi.

Kako bi algoritam simulacije kretanja mnoštva ljudi bio vjerodostojan, mora zadovoljavati određene uvjete. Primjerice, za simulacije evakuacije zgrade potrebno je da se jedinke kreću prema cilju, izbjegavaju sudare te pokazuju druga ljudska ponašanja.

Prvi algoritam ove vrste razvio je Craig Reynold 1987. godine. Njegov algoritam Boida, o kojemu se prethodno govorilo, tada se uključivao u grupu algoritama znanih kao bihevioralna animacija (engl. Behavioral animation). [7] Ovaj algoritam postavio je temelje simulacije gomila uvevši pojam agenata koji, prateći vrijednosti ostalih agenta, odlučuju o vlastitom kretanju.

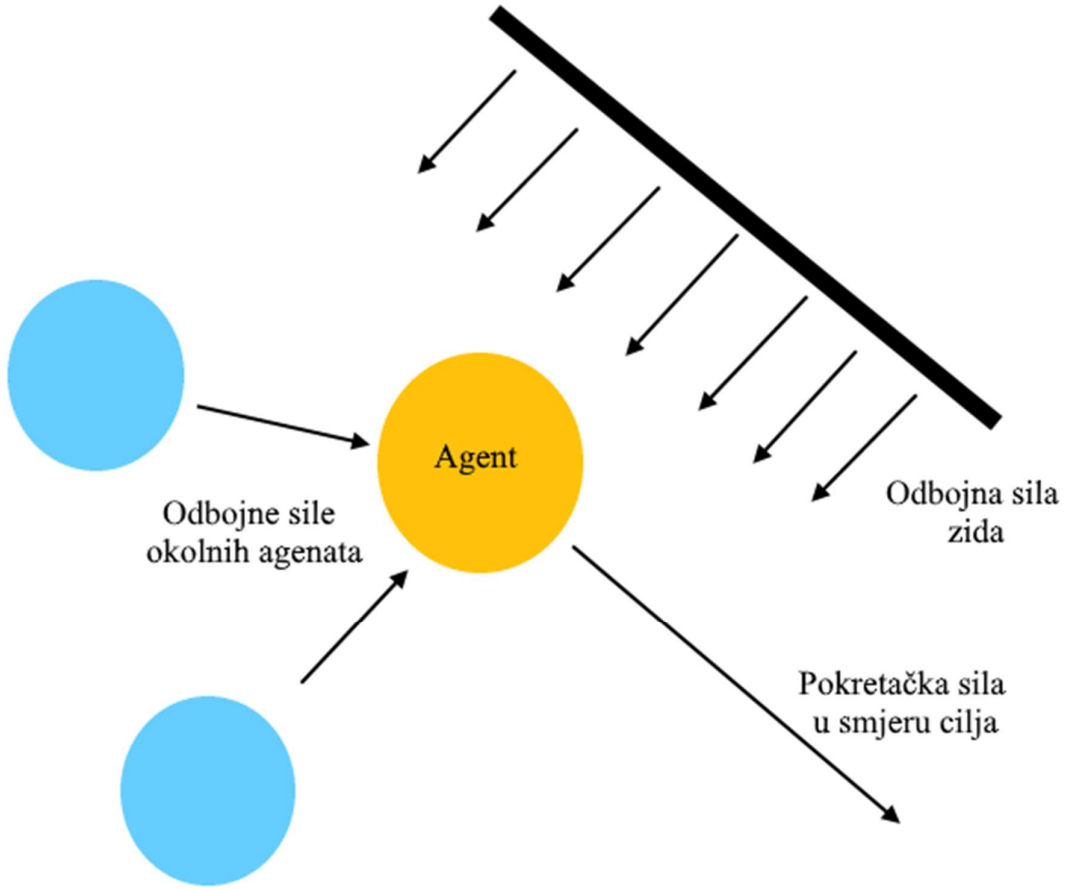
Daljnjim proučavanjem ovog algoritma razvilo se mnoštvo drugih tehnika koje se danas dijele u 3 kategorije ovisno o njihovom pristupu. Prvi se pristup fokusira na gomilu kao cjelinu i proučava njezin tok bez pridavanja pažnje pojedinim članovima. Ovaj pristup je ajčešće korišten za prikaz velikih skupina na kraćim intervalima. Drugi pristup uvodi subjekte koji i dalje nemaju sposobnost odlučivanje, već njihovim kretanjem upravljaju opći zakoni simulacije. Ovom pristupu najbolje odgovaraju male do srednje velike skupine s kratkoročnim ciljevima. Posljednji pristup uvodi pametne agente koji mogu donositi odluke o svom smjeru kretanja. Reagiraju na situacije na temelju više pravila odlučivanja te informacija prikupljenih iz svoje okoline. Najčešće se koristi za simulaciju realističnog kretanja gomila.

Ovaj će se rad pobliže baviti dvama algoritmima kretanja gomila. Prvi algoritam je model socijalne sile (engl. Social Force Model) u kojem postoje subjekti koji ne odlučuju samostalno. Drugi je algoritam uzajamnih prepreka brzine (engl. Reciprocal Velocity Obstacles) u kojemu agenti samostalno odlučuju.

4.2. Model socijalne sile (engl. Social force model)

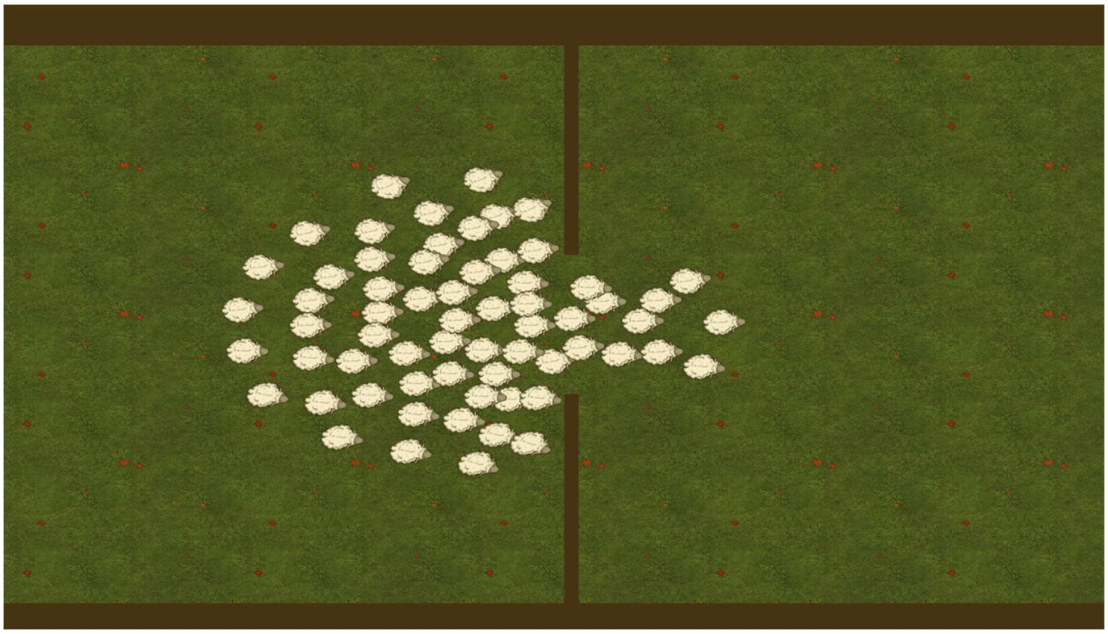
Model socijalne sile pogodan je za simulaciju gomila prilikom evakuacije ili drugim scenarijima u kojima je mnogo ljudi skučeno u malom prostoru. Nije pogodan za simulaciju gomila na otvorenim prostorima poput kretanja na pješačkim prijelazima i šetnicama.

Temeljna ideja algoritma je postojanje odbojnih, odnosno pokretačkih sila koje djeluju na svakog agenta. U svakom trenutku simulacije agenti izračunavaju željeni smjer kretanja na temelju djelujućih sila. Odbojne sile javljaju se kao posljedica zidova i prepreka te drugih agenta. Njihova je veličina obrnuto proporcionalna s kvadratom udaljenosti od agenta. Nasuprot odbojnih sila, pokretačka sila privlači agente prema njima zadanom cilju. Djelovanje navedenih sila prikazuje se na slici 4.1.

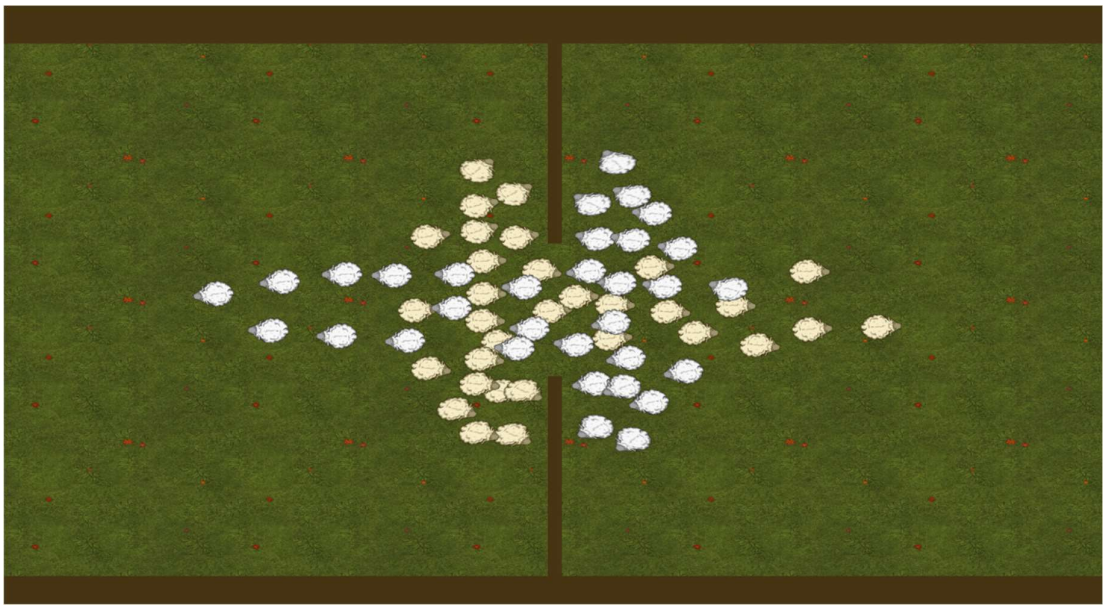


Slika 4.1. Dijagram sila modela

Prikaz simulacije pomoću ovog algoritma prikazan je na slikama 4.2. i 4.3.

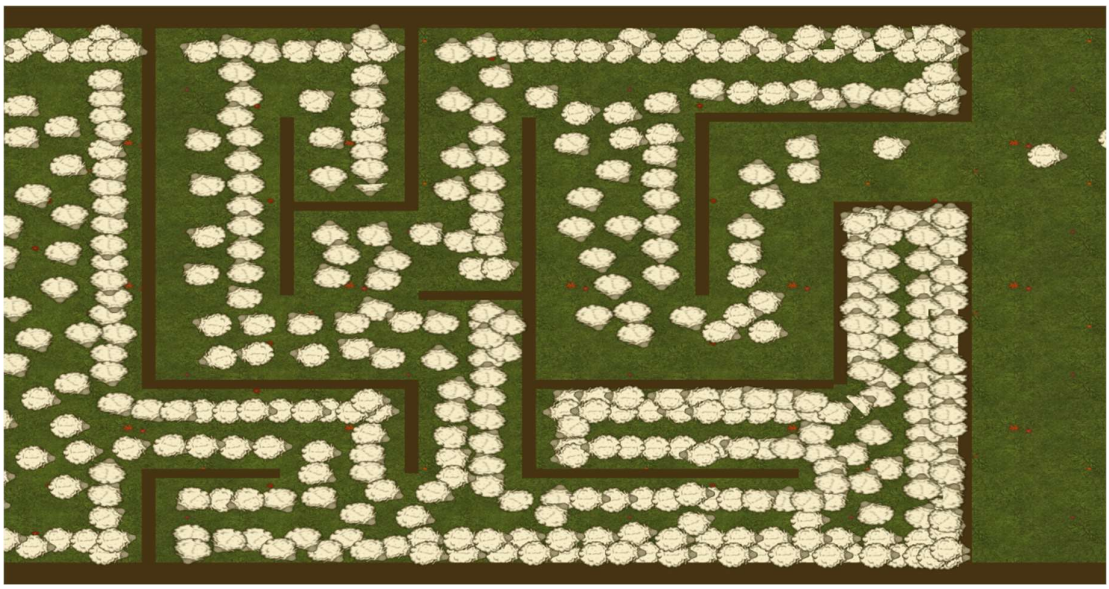


Slika 4.2. Simulacija s jednom grupom agenta



Slika 4.3. Simulacija s dvije skupine agenta

Funkcionalnost algoritma se smanjuje kako je tlocrt kompliciraniji. Primjer je vidljiv na slici 4.4. gdje je simulirano kretanje u labirintu. Zbog toga što agenti prate samo jedan fiksni cilj te nemaju nikakve dodatne uvjete, može se javiti situacija ekvilibrija što znači da neki ili svi agenti nikad neće stići na svoj cilj.

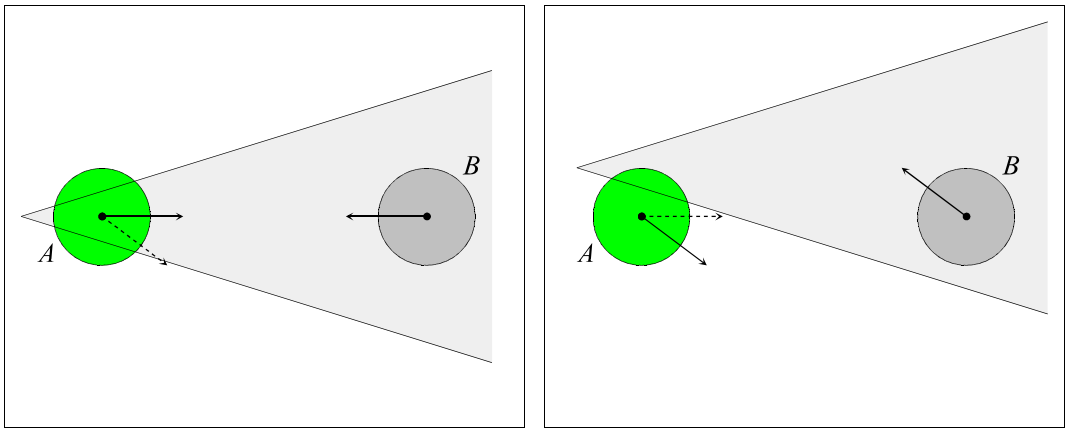


Slika 4.4. Simulacija kretanja u labirintu

Ostvarene performanse pri korištenju ovog algoritma su 750 do 900 sličica za 100 agenata, odnosno nešto manje od 100 sličica za 1000 agenta.

4.3. Algoritam uzajamnih prepreka brzine

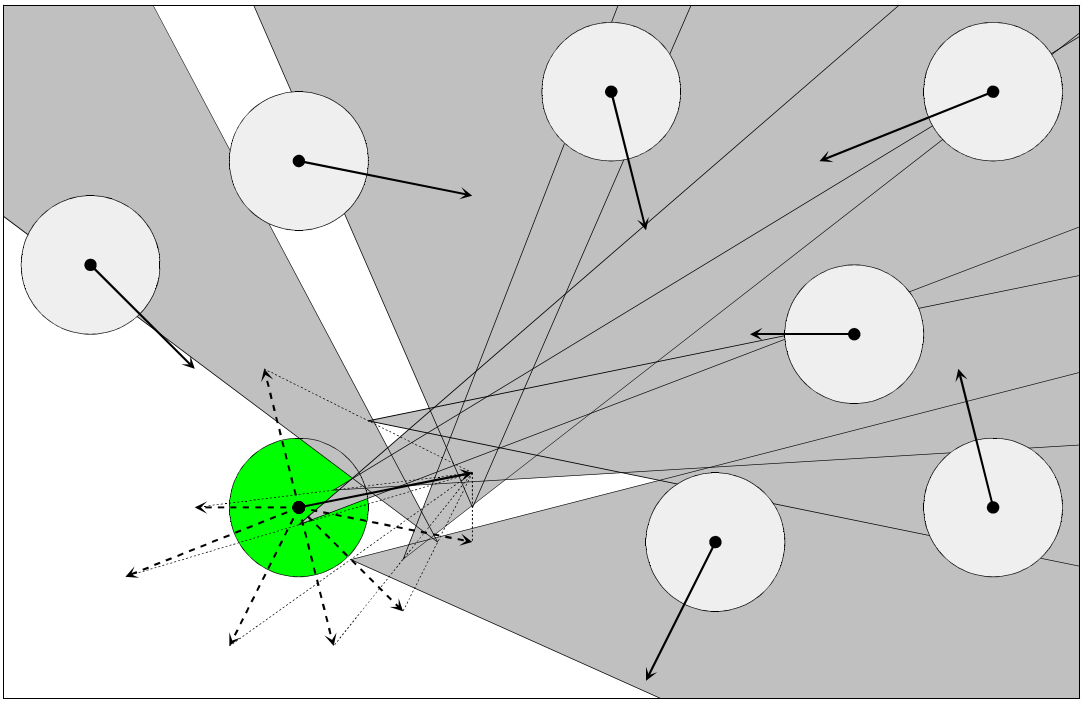
Algoritam uzajamnih prepreka brzine (engl. Reciprocal velocity obstacles) je jednostavan i brz algoritam za izbjegavanje sudara unutar gomila. Pogodan je za primjenu na otvorenim prostorima. Na početku simulacije svakom agentu zadaje se cilj. Agent se u svakom trenutku pokušava kretati prema tom cilju svojom željenom brzinom. Kako bi izbjegao sudare, agent, na slici 4.5. označen kao A, mora uzeti u obzir položaj, smjer kretanja i brzinu okolnih agenta. Za svakog obližnjeg agenta, poput onog na slici označenog kao B, konstruira se zona sudara pod pretpostavkom da B miruje. Nakon toga zona sudara se pomiče za prosječni vektor brzina agenata A i B kako je prikazano na slici 4.4. Cilj agenta A je da izabere novi smjer kretanja koji ne rezultira sudarom, a što bliže svojem željenom smjeru kretanja.



Slika 4.5. Prikaz određivanja zone sudara (lijevo) te njen pomak za prosječni vektor brzine (desno)

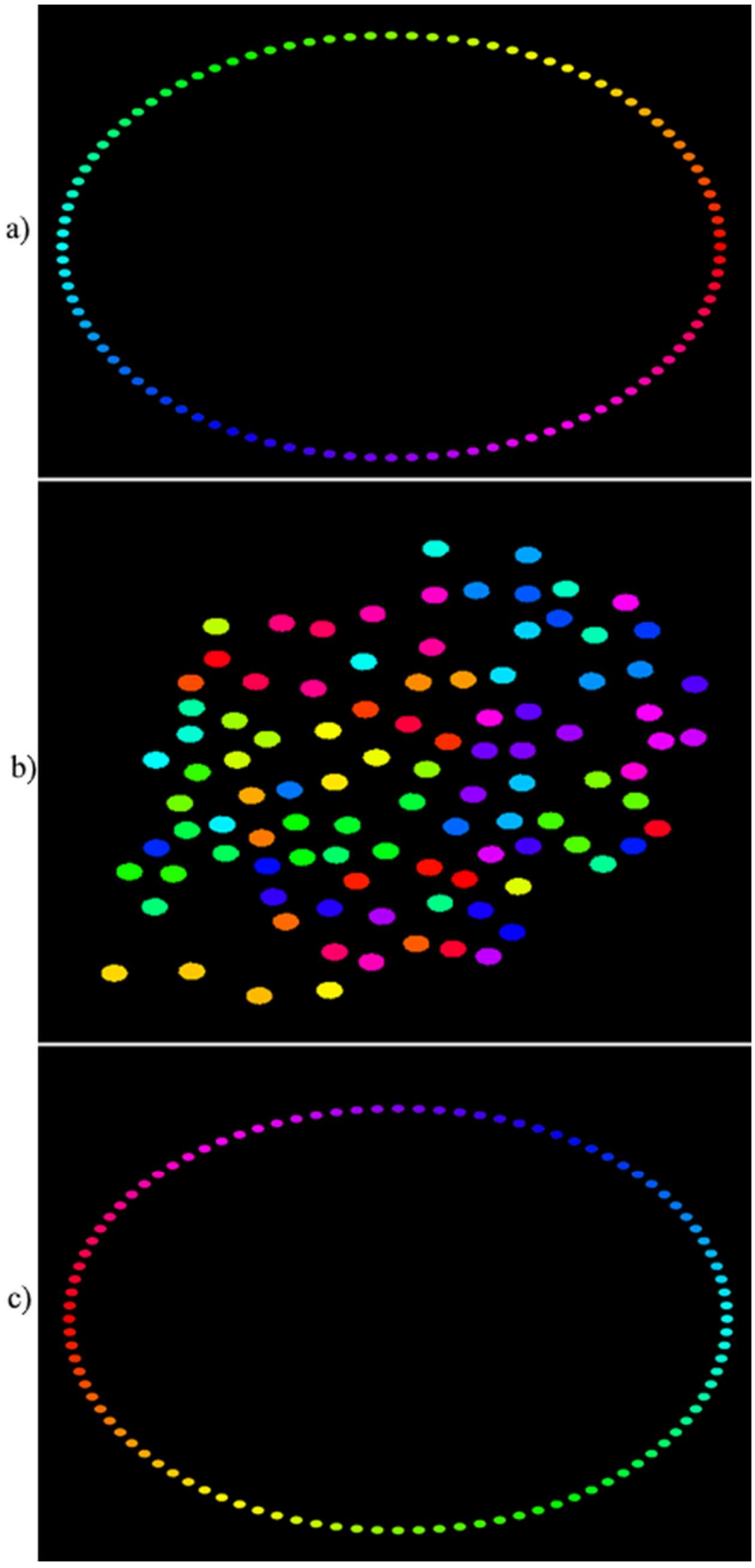
Isti postupak se izvodi za sve obližnje agente te se dobiva unija zona sudara kao što je prikazana na slici 4.6. Agent izabire novi smjer kretanja koji nije u uniji, a što je bliže

njegovom željenom smjeru kretanja. Ukoliko simulacija uključuje prepreke, zone sudara se također moraju izračunati i za njih, a ne samo za obližnje agente.



Slika 4.6. Određivanje zona sudara u kompleksnom scenariju

Na slici 4.7. prikazana je simulacija pomoću algoritma uzajamnih prepreka brzine.



Slika 4.7. Simulacija kretanja, a) Početno stanje, b) Stanje na sredini, c) krajnje stanje

Zaključak

Algoritmi poput roja čestica pružaju jedinstvene načine za generiranje realističnih i kompleksnih oblika pri čemu se zaobilazi potreba za dugotrajnim programiranjem mnoštva detalja.

Simulacije gomila od velike su važnosti u filmskoj industriji te industriji video igara. Omogućavaju jednostavan i brz prikaz velikih skupina jedinki koje se kreću i ponašaju inteligentno. Od velike je važnosti da ovi algoritmi budu optimizirani kako bi se u sustavima koji su predviđeni za rad u realnom vremenu ostvarile što bolje performanse.

Osim korištenja u zabavnoj industriji, donose mnoge napretke u urbanom planiranju te planiranju kriznih situacija. Budući da se njihovom uporabom mogu dobiti veoma realistična predviđanja ljudskog ponašanja, omogućavaju bolju i detaljniju izvedbu prometnih rješenja te planova evakuacije.

Literatura

[1] https://github.com/TinSrnic/diplomski-rad

Accessed on 2021-06-20. 2021.

[2] Gerardo Beni, Jing Wang. *Swarm Intelligence in Cellular Robotic Systems*. 1993

[3] *Conway's Game of Life*.
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
Accessed on 2021-05-02. 2021.

[4] Rendell, Paul. *Turing machine universality of the game of life.* 2016

[5] Craig, Reynolds. *Flocks, Herds, and Schools: A Distributed Behavioral Model.* 1987

[6] Jones, Jeff. *Characteristics of pattern formation and evolution in approximations of physarum transport networks.* 2010

[7] Craig Reynolds. *Steering Behaviors For Autonomous Characters.* 2002

Vizualizacija simulacije kretanja mnoštva ljudi

Sažetak

Ovaj se rad bavi problematikom simulacije kretanja gomila. Opisuje se nekolicina algoritama inteligencije roja poput Boida. Fokus se stavlja na algoritme kretanja ljudskih gomila te se pobliže opisuje model socijalne sile te algoritam uzajamnih prepreka brzine. Sve simulacije napravljene su u programskom jeziku C++ koristeći OpenGL zajedno sa programskim bibliotekama GLFW i GLEW. Korištena je grafička kartica Nvidia GTX 770 i procesor Intel Core i7-4771.

Ključne riječi: C++, OpenGL, Inteligencija roja, kretanje gomila

Visualization of Movement in Crowd Simulations

Summary

This paper deals with the problem of crowd motion simulation. Several swarm intelligence algorithms such as the Boid algortihm are described. It focuses on movement algorithms for human crowds such as the Social Force Model and Reciprocal Velocity Obstacles algorithm. All simulations were made in the C ++ programming language using OpenGL together with the GLFW and GLEW programming libraries. All tests used the Nvidia GTX 770 graphics card and the Intel Core i7-4771 processor.

Ključne riječi: C++, OpenGL, Swarm intelligence, movement of crowds