SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Višeagentni sustav unutar jata dronova

Projekt

Paula Kokić
br. indeksa: 45263/16-R

Mentor:
Prof. dr. sc. Markus Schatten

Varaždin, 4. veljače 2018.

Sadržaj

Poglavlje 1

Uvod

U ovom radu bavit ću se opisom i rješavanjem problema samoorganizacije dronova u jatu. Dronovi (eng. *drones*) ili autonomne (bespilotne) letjelice su letjelice bez posade, odnosno pilota koji leti s njim. Umjesto toga, njom se može upravljati na daljinu, dok neke imaju čak sposobnost samostalnog leta koji je unaprijed isprogramiran ili se generira dinamički. Koriste se u vrlo različite civilne ili vojne svrhe, a nerijetko u jatu, odnosno kao skupina. (TheUAV.com, 2016)

Jedan od načina za lakše upravljanje jatom dronova koje se kreće jest da se svi članovi jata poslože u određenu zadanu formaciju letenja s određenom udaljenošću, ovisno o broju članova i ostalim parametrima. Koristeći koncepte iz samoorganizacije (eng. *self-organization*) pokušat ću napraviti algoritam kojim će se jato dronova organizirati u određenu formaciju te zajedno nastaviti kretanje. Samoorganizacija predstavlja sposobnost nekog sustava da funkcionira bez posredovanja centralne kontrole. To znači da inicijativa i ponašanje svakog pojedinačnog člana unutar sustava kombinira kako bi se dobio konačan rezultat. (Agent.GUI, 2018)

Poglavlje 2

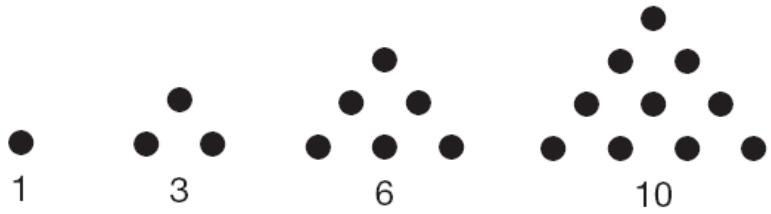
Opis problema i ciljeva

Kako bi problem bio lakše opisan, razložit ću ga na tri dijela.

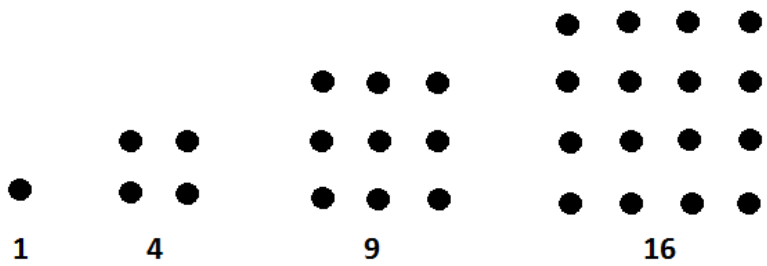
Prvi dio problema odnosi se na sam protokol samoorganizacije dronova. Kako započeti inicijativu samoorganizacije? Kako ćemo odrediti kada su dronovi izvršili zadanu naredbu?

Drugi dio problema odnosi se na implementaciju agenta, odnosno komunikaciju između agenata. Koje poruke i podatke moraju međusobno razmjenjivati kako bi uspješno izvršili zadanu naredbu?

Treći dio odnosi se na samu definiciju formacije letenja. Pod pojmom "format" ili "formacija" letenja, smatra se određeni geometrijski oblik (npr. kvadrat, trokut), odnosno svi geometrijski oblici/veličine određenog figurativnog broja. Figurativni brojevi su prirodni brojevi koji se pomoću točkica mogu poslagati u određene geometrijske oblike. Prema tome postoje trokutasti (slažu se u trokut), kvadratni (slažu se u kvadrat), produljeni (slažu se u pravokutnik), itd. figurativni brojevi. Primjer trokutastih i figurativnih brojeva možemo vidjeti na slikama 2.1 i 2.2.



Slika 2.1: Trokutasti brojevi



Slika 2.2: Kvadratni brojevi

U našem slučaju, Svaka točkica na slici predstavlja jednu poziciju u formatu letenja. Za odabrani format (trokut, kvadrat, itd.) uzima se prvi veći figurativni broj od broja dronova u jatu - naravno, neka mjesta će ostati upražnjena, ali to nam u ovoj domeni ne predstavlja problem. Problem predstavlja upravo računanje ovih pozicija, odnosno lokacija u koordinatnom prostoru, kako bi dronovi znali kuda se trebaju kretati.

Cilj ovog projekta jest osmisliti algoritam, tj. protokol komunikacije među čvorovima unutar jata dronova. Svaki čvor bit će jedan agent te kao član jata, ujedno i član višeagentnog sustava. Rezultat algoritma bi trebali biti čvorovi posloženi u zadani format te takvi zajedno nastaviti svoje kretanje. Što se tiče formacije letenja, zbog jednostavnosti izračuna, u ovom radu zadržat ćemo se na rješenju kvadratnog formata letenja.

Rješenje će biti implementirano u `C#` programskom jeziku, odnosno u .NET tehnologiji, uz pomoć Visual Studio 2015 razvojnog okruženja.

Prije nego krenem u prijedlog rješenja, u sljedećem poglavlju bit će opisana neka od sličnih istraživanja na ovu temu.

Poglavlje 3

Slična istraživanja

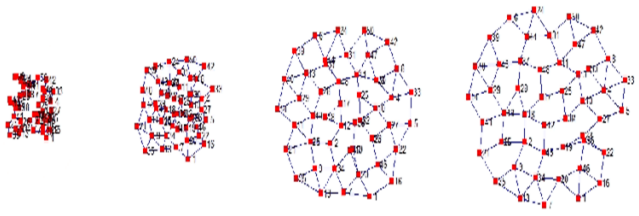
U ovom poglavlju bit će ukratko opisana neka slična istraživanja povezana s ovom temom.

3.1 Jeong D., Lee K. (2014) Dispersion and Line Formation in Artificial Swarm Intelligence

Dvije autorice sa Sveučilišta *Case Western Reserve University* bave se dinamičkim modelom kolektivnog sustava. Napominju kako je jedna od najvažnijih stvari u inteligenciji jata da unatoč tome što individue slijede jednostavna pravila, rezultirajuće globalno ponašanje može biti inteligentno i kompleksno. Riječ je o tome da se mali dijelovi sustava koordiniraju tako da uspiju riješiti kompleksne zadatke koje nisu sposobni riješiti sami. Takvi sustavi mogu funkcionirati na lokalnoj razini - kada individue informacije dijele sa svojim susjedima, tj. u određenom dosegu ili na globalnoj razini, kada rojem centralizirano upravlja jedan vođa. Globalna formacija nekog oblika je jedna od ozbiljnih problema inteligencije rojeva, a u prirodi se koristi u različite svrhe (kod ptica, mrava i sl.)

Njihov dinamički model sustava jata kojeg sačinjavaju agenti bazira se na virtualnom mehanizmu opruge i prigušivača, kako bi se što bolje opisalo međusobno udaljavanje agenata. Pretpostavka je da se radi o dvodimenzionalnom prostoru te je korištena formula za udaljenosti dvije točke u koordinatnom sustavu za izračun udaljenosti između dva agenta. Korištena je i formula drugog Newtonovog zakona, kako bi se opisala sila koja gura pojedini agent u određenom smjeru, ovisno o privlačnim i odbojnim silama njegovih susjeda. Podrazumijevano je da agent pozna samo udaljenosti do susjednih agenata.

Opisana su dva algoritma: disperzni, baziran na triangularnim planarnim elementima bez vođe i linijska formacija koristeći uparene linijske elemente. Provedena je simulacija u MATLAB-u s 50-ak agenata s različitim komunikacijskim dometom za agente. Time je dokazana valjanost danih formula za disperziju. Rezultati simulacije vidljivi su na slici 3.1. (Jeong and Lee, 2014)



Slika 3.1: Disperzija (Jeong and Lee, 2014)

3.2 Hexmoor H., McLaughlan B., Baker M. (2005): Swarm Control in Unmanned Aerial Vehicles

Trojica autora sa sveučilišta University of Arkansas bavila su se modelom kortrole flote dronova uz minimalnu korisničku intervenciju. Cilj flote dronova je skenirati i prikupiti informacije o određenoj meti na zemlji.

Radi se o dvodimenzionalnoj simulaciji, implementiranoj u Javi, gdje se dronovi i mete kreću po pretpostavljenoj mreži. Kada se meta (vozilo) skenira dulje od 10 vremenskih jedinica, smatra se da je riješena i može se tražiti dalje. Ako se izgubi nadzor nad pojedinom metom, moguće je kasnije obnoviti vidljivost i nastaviti skeniranje. Dronovi (agenti) mogu međusobno komunicirati i pregovarati o budućim metama pa prema tome odustati od jedne mete i krenuti prema drugoj na nagovor susjednog agenta. Osim toga, mogu biti i sugerirani od strane čovjeka, koji može određeni prostor označiti kao "hladno područje" - za koje smatra da je manja šansa da postoji meta ili "toplo područje" - za koje smatra da je veća šansa da postoji meta. Agenti također imaju i parametriziranu osobnost, a karakteristike su: konformizam (eng. *conformity*), socijabilnost (eng. *sociability*), upornost (eng. *dedication*, koliko brzo agent odustaje nakon što izgubi metu iz vida) i narav (eng. *disposition*). Korisnik u svakom trenutku može mijenjati ova četiri parametra te se ona primjenjuju na dronove koji se trenutno nalaze na polju.

Konformizam se odnosi na to koliko agent uzima u obzir korisnikovu sugestiju toplih i hladnih područja. Npr. agent s malo konformizma neće puno uzimati u obzir korisnikove sugestije. Ukoliko je ova vrijednost na nuli, skroz se poništava efekt korisnikove sugestije i zapravo je kontraproduktivno. Teoretski, moguće ju je postaviti i na negativnu vrijednost, kada agent zapravo radi suprotno od onoga što korisnik sugerira.

Socijabilnost određuje koliko će neki dron biti društven. Onaj s većom socijabilnošću će pokušati surađivati s ostalim dronovima, dok onaj s manjom će pokušati izbjegavati druge dronove i raditi samostalno. Kada se razmatra određena destinacija, uzima se u obzir prisutnost drugih dronova te se ponaša u skladu s time. Kada je ova vrijednost prevelika, postaje kontraproduktivna, budući da će dronovi uvijek ići jedan k drugome, ostajati u grupama i time gubiti mogućnost istraživanja cijelog područja jer neće nikuda otići sami.

Upornost određuje kolika je želja dronova da pronađu izgubljenu metu. Dron s malom upornošću brzo će odustati nakon što izgubi metu iz vidnog polja, dok će onaj s višom biti uporniji. Kada dron odluči ne tražiti izgubljenu metu, obznanjuje ostalima da je lokacija te mete izgubljena i potražit će novu metu. Dron koji odluči ganjati svoju metu, neće poslati poruku da je meta izgubljena, umjesto toga će ostalima obznaniti posljednju poznatu lokaciju dok god ili ponovno ne uhvati metu ili neki drugi dron ne locira metu i obznani njezinu lokaciju ili konačno ne odustane i odluči se za novi plan akcija.

Narav je mjera za frustraciju tijekom pregovaranja s ostalim dronovima, odnosno broj vremenskih jedinica koliko će ostati frustriran bez odustajanja od mete.

Tijekom odlučivanja o odabiru mete, svaki dron predlaže svoju metu. Ako dvojica predlože istu, pobjeđuje onaj koji je bliži. Oni koji izgube, moraju dalje sami lutati, do sljedeće faze pregovora.

U konačnici, ovaj rad pomiče granice kontrole dronova od klasičnog, striktnog daljinskog upravljača prema inteligentnim, cilju-usmjerenim, kooperativnim agentima. Korisnik i dalje ima veliku ulogu u kontroli jata, međutim ipak je puno jednostavnije od starijih simulacijskih modela. (Baker et al., 2005)

Poglavlje 4

Model drona

Prije samog opisa rješenja, potrebno je definirati i pojasniti određene pretpostavke. U sustavu jata dronova koristit će se definirani model drona kako slijedi.

Svaki dron predstavlja jedan čvor, odnosno agent sa svojim svojstvima i ponašanjima. Svi dronovi pripadaju istoj klasi, bez obzira na buduću ulogu u jatu i imaju jednake sposobnosti. Svojstva svakog drona su: Naziv, Početna lokacija (x,y), smjer, brzina kretanja (broj mjernih jedinica udaljenosti u sekundi). Pretpostavljene sposobnosti i ponašanja dronova su: kretanje u dvodimenzionalnom prostoru, opremljenost određenim GPS prijemnicima za određivanje lokacije te WiFi odašiljačima za razmjenu podataka. Budući se bavimo malim područjem, pretpostavljamo da svi dronovi vide (očitavaju) sve ostale dronove te mogu primati svačije poruke i svima poslati poruke. Nemaju globalno znanje, već svaki dron za sebe nakon primitka određene poruke ažurira ono što je potrebno. Ono što dronove međusobno razlikuje su specifične vrijednosti određenih svojstava, kao što je početna lokacija, smjer i brzina te naziv.

U implementaciji, jedan dio simulatora je već napravljen u sklopu jednog drugog projekta te će ovo biti nadogradnja na isti. Prema tome, pretpostavljamo da je već implementirano kretanje dronova u dvodimenzionalnom prostoru, koje se zapravo zasniva na najobičnijoj formuli za brzinu, uz dodatak izračuna potrebnih smjerova kako bi se mogla odrediti konačna lokacija drona. Ono što će biti objašnjeno u ovom radu jest protokol za komunikaciju između dronova koji obuhvaća način razmjene informacija među dronovima u jatu, implementaciju samih agenata i njihovih ponašanja te implementaciju izračuna samog oblika formacije.

Poglavlje 5

Opis rješenja

U ovom poglavlju bit će redom opisano rješenje, prema postavljenim pitanjima kod definicije problema: algoritam samoorganizacije, implementacija agenata te implementacija izračuna formata letenja.

5.1 Algoritam samoorganizacije

Ovdje je važno napomenuti da se ne radi o potpunoj samoorganizaciji, niti pravoj inteligenciji rojeva budući da ćemo uvesti sljedeće pretpostavke: jato ima vođu koji će započeti inicijativu organizacije, i svaki član jata zna sve što znaju i drugi članovi jata. Da se radi o potpunoj samoorganizaciji ili inteligenciji roja, ne bi bilo potrebe za vođom i svaki član bi samo znao lokalne informacije, tj. na primjer udaljenosti do i lokacije susjednih čvorova.

Koraci algoritma su sljedeći:

1. **Odredi vođu jata.** Kao vođu se postavlja prvi definirani član u jatu. U budućim razmatranjima može se razviti protokol dogovora među dronovima tko će biti vođa.
2. **Vođa jata generira lokacije.** Vođa zna koliko ima članova u jatu. Na temelju tog broja generira potreban broj lokacija s obzirom na odabranu formaciju. S obzirom na svoju trenutnu lokaciju i trenutni smjer kretanja, prema odabranom formatu letenja generira točne koordinate svake lokacije.
3. **Vođa šalje zahtjev za organizacijom.** Svakom članu jata šalje se obavijest da je potrebno provesti organizaciju. Skupa s tom obavijesti, vođa šalje sve generirane (tj. moguće) lokacije, zadani smjer i brzinu kretanja te svoju lokaciju.
4. **Član jata prima obavijest od vođe.** Sprema primljene podatke. Od ponuđenih lokacija, izabire onu koja mu je najbliža te šalje svim ostalim članovima svoju odabranu lokaciju, tako da znaju da je ta lokacija zauzeta.
5. **Vođa prima lokacije i šalje zahtjev za kretnjom.** Kao i svi ostali članovi jata, vođa također prima lokacije dronova koji su odlučili o svojim lokacijama. Kada primi sve lokacije, vođa šalje znak da svi mogu krenuti na svoje pozicije, tj. lokacije.
6. **Članovi jata primaju znak za kretnjom.** Nakon primljene obavijesti, kreće prema svojoj lokaciji. Kada stigne na lokaciju, šalje poruku vođi da je stigao.

7. **Vođa prima obavijesti o dolasku na lokacije.** Kada vođa zaprimi obavijesti o dolasku svih članova jata na svoje lokacije, šalje novu poruku svim članovima za nastavkom kretanja.

5.2 Implementacija agenata

U nastavku slijedi opis ponašanja i funkcionalnosti agenata. Bit će objašnjeni i pojedini načini izračuna određenih vrijednosti, te prikazan originalni programski kôd. Budući da je ovaj projekt zapravo proširenje mog završnog rada, u kojem imam već definirano i implementirano kretanje dronova, ovdje će biti prikazana samo ponašanja agenata, što je i tema ovog rada.

5.2.1 Član

Svi dronovi, članovi jata i njihov vođa, imaju zajednička ponašanja i instance su iste klase *Dron*. Jedina je razlika u tome što vođa primjenjuje neka dodatna ponašanja, koja će biti objašnjenja u sljedećem potpoglavlju. Dakle, svaki dron ima određena svojstva koja ga određuju: njegova početna lokacija (x,y), početni smjer, trenutna lokacija, trenutni smjer i brzina. Za potrebe aplikacije, dodani su atributi naziva i boje kako bi se dronovi prema potrebi mogli razlikovati.

Prva aktivnost koja se provodi od pojedinog člana jata jest primanje obavijesti od vođe da se treba organizirati. Za to se brine metoda *PrimiPodatke()* koja je prikazana ispod:

```
public void PrimiPodatke(List<MyPoint> lista, float smjer, float brzina)
{
        generiraneLokacije = new List<MyPoint>(lista);
        buduciSmjer = smjer;
        buducaBrzina = brzina;
}
```

Nakon primanja i spremanja podataka o generiranim lokacijama, smjeru i brzini budućeg kretanja, prima i lokaciju na kojoj se nalazi vođa. Pomoću metode PrimiLokaciju() prima lokaciju od bilo kojeg drugog drona, ne samo od vođe. Primljena lokacija sprema se u listu zauzetih lokacija:

```
public void PrimiLokaciju(MyPoint lokacija)
{
        primljeneZauzeteLokacije.Add(lokacija);
}
```

Nakon primitka svih potrebnih podataka, član na temelju svoje trenutne lokacije pronalazi najbližu slobodnu lokaciju od ponuđenih (generiranih) lokacija koje mu je vođa poslao. Za to se brine metoda *IzaberiNajblizuLokaciju()*:

```
public void IzaberiNajblizuLokaciju()
{
        double minUdaljenost = double.MaxValue;
        double udaljenost;
        MyPoint najblizaLokacija = new MyPoint();
        foreach (MyPoint p in generiraneLokacije)
        {
                MyPoint postojeca = primljeneZauzeteLokacije.Find(tocka =>
                    Math.Round(tocka.X, 1) == Math.Round(p.X, 1) &&
                    Math.Round(tocka.Y, 1) == Math.Round(p.Y, 1));
```

```
                if (postojeca == null)
                {
                        udaljenost = Math.Sqrt(Math.Pow((TrenX - p.X), 2)
                            + Math.Pow((TrenY - p.Y), 2));
                        if (udaljenost < minUdaljenost)
                        {
                                minUdaljenost = udaljenost;
                                najblizaLokacija = p;
                        }
                }
        }
        buducaLokacija = new MyPoint(najblizaLokacija.X,
            najblizaLokacija.Y);
        PosaljiLokaciju();
}
```

Kao što možemo vidjeti, agent pomoću formule za udaljenost dviju točaka u koordinatnom sustavu izračunava udaljenost do svake točke koja nije već zauzeta te u konačnici za svoju buduću lokaciju sprema onu s najmanjom udaljenošću. Na kraju s metodom *PosaljiLokaciju()* šalje svoju buduću lokaciju svim ostalim čvorovima:

```
public void PosaljiLokaciju()
{
        foreach (Dron d in Dron.ListaDronova)
        {
                d.PrimiLokaciju(buducaLokacija);
        }
}
```

U konačnici, kada svi članovi jata odaberu svoje lokacije, primaju zahtjev od vođe da krenu prema svojoj lokaciji. Za primanje tog znaka služi metoda *IdiPremaLokaciji()*:

```
public void IdiPremaLokaciji()
{
        if (!poslano) this.TrenSmjer = izracunajSmjerPremaLokaciji();
        if (Math.Round(TrenX, 0) != Math.Round(buducaLokacija.X, 0) &&
            Math.Round(TrenY, 0) != Math.Round(buducaLokacija.Y, 0))
        {
                this.PomakniDron();
                double udaljenost = Math.Sqrt(Math.Pow((TrenX -
                    buducaLokacija.X), 2) + Math.Pow((TrenY -
                    buducaLokacija.Y), 2));
                if (udaljenost < Brzina) Brzina = 1;
        }
        else
        {
                if (!poslano)
                {
                        Dron vodja = ListaDronova.ElementAt(0);
                        vodja.PrimiObavijestODolasku();
```

```
                        poslano = true;
                        this.TrenSmjer = buduciSmjer;
                        this.Brzina = buducaBrzina;
                }

        }
}
```

Vidimo da se po dolasku na lokaciju šalje informacija vođi jata. U sljedećem poglavlju vidjet ćemo kako ovi procesi izgledaju iz perspektive vođe dronova.

5.2.2 Vođa

Krajnji korisnik sustava zapravo započinje inicijativu organiziranja svojeg jata. Na znak korisnika, odabire se vođa jata i pokreće se samoorganizacija dronova u jatu. Osnovna metoda od koje sve kreće je *pokreniOrganizaciju()*:

```
private void pokreniOrganizaciju()
{
        this.generirajLokacije();
        Brojac = 0;
        foreach (Dron d in ListaDronova)
        {
                if (d != this)
                {
                        d.PrimiPodatke(form.ListaPozicija, TrenSmjer,
                            Brzina);
                        d.PrimiLokaciju(form.ListaPozicija.ElementAt(0));
                }
        }
        foreach (Dron d in ListaDronova)
        {
                if (d != this)
                {
                        d.IzaberiNajblizuLokaciju();
                }
        }
}
```

Vidimo da se unutar ove metode događaju skoro sve akcije koje vođa provodi. Prva od tih akcija je generiranje lokacija. U metodi *generirajLokacije()* pozivaju se druge odgovarajuće metode za matematički izračun svih lokacija s obzirom na broj čvorova u jatu. Nakon toga vidimo da se svim dronovima šalje lista generiranih pozicija, smjer kojim se trebaju gibati te brzinia. Isto tako, šalje se i lokacija na kojoj se nalazi sam vođa. Nakon toga vidimo da nakon što su svi dronovi primili podatke, počinje izračun njihovih najbližih lokacija. Kad neki dron izračuna svoju buduću lokaciju, šalje ju svim dronovima. Nakon što su svi dronovi izabrali svoje lokacije, šalje se zahtjev za kretnjom prema odabranoj lokaciji, odnosno *IdiPremaLokaciji()* koja je već pokazana gore. Kada određeni dron stigne na svoju lokaciju, obavještava vođu o tome pozivajući metodu *PrimiObavijestODolasku()*:

```
public void PrimiObavijestODolasku()
{
        Brojac++;
}
```

Vođa zapravo samo broji koliko je obavijesti primio i kad je primio točan broj obavijesti, izdaje konačan zahtjev za kretanjem svih dronova odjednom u zadanom smjeru i zadanoj brzini.

5.3 Implementacija izračuna formacije

Za potrebe računanja svih potrebnih lokacija određene formacije napravljena je zasebna klasa *Formacija*. U pravilu, njen objekt instancira i koristi samo vođa kako bi mogao generirati lokacije i poslati ostalim članovima jata. Klasa sadrži svojstva *BrojCvorova, BrojRedova, BrojStupaca, SmjerKretanja, Razmak, Brzina, SmjerPoretka* te listu *ListaPozicija*. Prilikom kreiranja samog objekta većina svojstava se automatski računa, kao što je prikazano:

```
public Formacija(int brojCvorova, float smjer, int razmak, float brzina)
{
        ListaPozicija = new List<MyPoint>();
        BrojCvorova = brojCvorova;
        BrojStupaca = IzracunajBrojStupaca();
        BrojRedova = IzracunajBrojRedova();
        SmjerKretanja = smjer;
        Razmak = razmak;
        Brzina = brzina;
        SmjerPoretka = IzracunajSmjerPoretka();
}
```

Broj redova i broj stupaca u ovom slučaju, kada imamo kvadratnu formaciju, računamo pomoću korijena. Prvo se računa broj stupaca: vadi se korijen od ukupnog broja čvorova. Ako je taj broj cijeli, onda je točno to i broj stupaca, u suprotnom, uzet ćemo prvi veći cijeli broj od tog broja. Implementacija je sljedeća:

```
public int IzracunajBrojStupaca()
{
        double korijen = Math.Sqrt(BrojCvorova);
        if (JelCijeliBroj(korijen))
        {
                return (int)korijen;
        }

        return (int)Math.Truncate(korijen) + 1;
}
```

Broj redaka je malo više varijabilan. Prvi dio računa je zapravo isti - ukoliko je korijen broja čvorova cijeli broj - to je broj redaka - i u tom slučaju imamo zapravo pravilan n x n kvadrat. Ukoliko nije, trebamo broj čvorova podijeliti s brojem dobivenih stupaca. Opet, ako je dobiven cijeli broj, to će biti broj redaka, u suprotnom, treba uzeti prvi veći cijeli broj. Implementacija je sljedeća:

```
public int IzracunajBrojRedova()
{
        double korijen = Math.Sqrt(BrojCvorova);
        if (JelCijeliBroj(korijen))
        {
                return (int)korijen;
        }
        double pom = BrojCvorova / (double)BrojStupaca;
        if (JelCijeliBroj(pom))
        {
                return (int)pom;
        }
        return (int)Math.Truncate(pom) + 1;
}
```

IzracunajSmjerPoretka() je metoda koja se bavi izračunom smjera u kojem idu pozicije. Naime, običan smjer kretanja je jedno, ali smjer poretka u kvadratnoj fomaciji onda je za 90 stupnjeva zakrenut u jednu stranu:

```
public float IzracunajSmjerPoretka()
{
        float s = SmjerKretanja + 90;
        if (s == 360) s = 0;
        else if (s > 360) s = s - 360;
        return s;
}
```

Naime, to je potrebno napraviti, jer se zapravo izračun točnih lokacija zasniva na istoj formuli koja inače i pomiče dronove u određenom smjeru - umjesto smjera kretanja prosljeđuje se smjer poretka, a umjesto brzine se prosljeđuje razmak koji treba biti između dronova. Implementacija konačne metode za izračun lokacija je sljedeći:

```
public void IzracunajPozicije(float x, float y)
{
        MyPoint prva = new MyPoint(x, y);
        ListaPozicija.Add(prva);
        MyPoint trenutna = prva;
        for (int i = 0; i < BrojRedova; i++)
        {
                for (int j = 0; j < BrojStupaca-1; j++)
                {
                        MyPoint iduca = Kretanje.NapraviPomak(trenutna,
                            SmjerPoretka, Razmak, false);
                        ListaPozicija.Add(iduca);
                        trenutna = iduca;
                }
                if (i < BrojRedova - 1)
                {
                        float s = SmjerPoretka + 90;
                        float b = (i + 1) * Razmak;
```

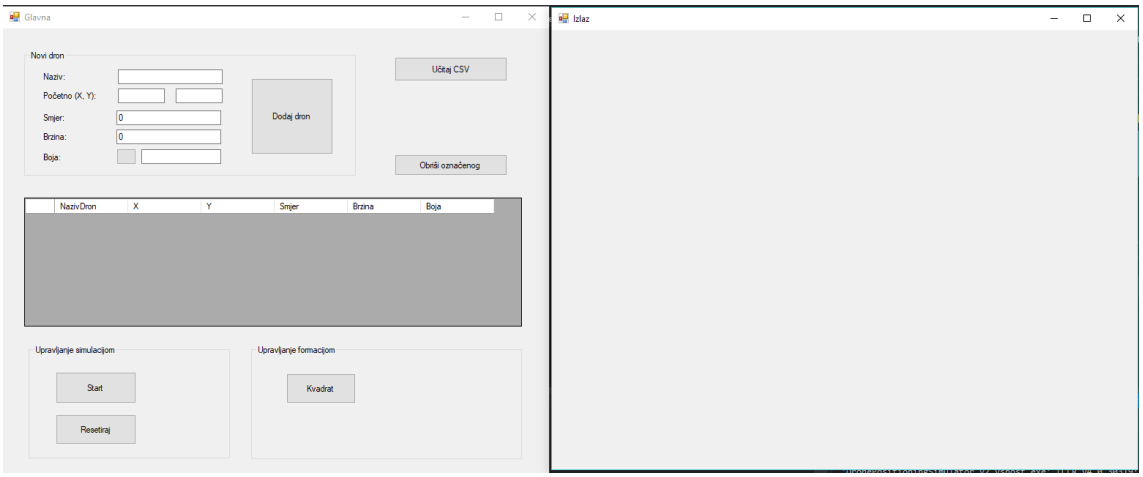
```
                        trenutna = Kretanje.NapraviPomak(prva, s, b,
                            false);
                        ListaPozicija.Add(trenutna);
                }
        }
}
```

Dakle, naprije se daje prva točka, tj. lokacija, a to je lokacija na kojoj se nalazi vođa, i od nje se kreće. Vanjska petlja broji redove, unutarnja stupce, tj. popunjava se izgled figurativnog broja red po red. Kao što je već napomenuto, iduća točka računa se uz pomoć prethodno definiranih metoda za kretanje samih dronova. Nakon određivanje iduće lokacije, ista se sprema u listu i postavlja kao trenutna, u odnosu na koju će se računati iduća. Petlja se vrti za jedan manje od broja stupaca, jer smo u onom prvom redu već dodali prvu točku. Zato da nadoknadimo tu jednu točku, prije nego se krene u nove iteracije, potrebno je izračunati idući prvu lokaciju u novom redu - i to se naravno radi samo ako se ne nalazimo u posljednjem redu. Ta lokacija se uvijek računa u odnosu na prvu, samo se razmak u odnosu na tu točku povećava.

Poglavlje 6

Primjer izvođenja

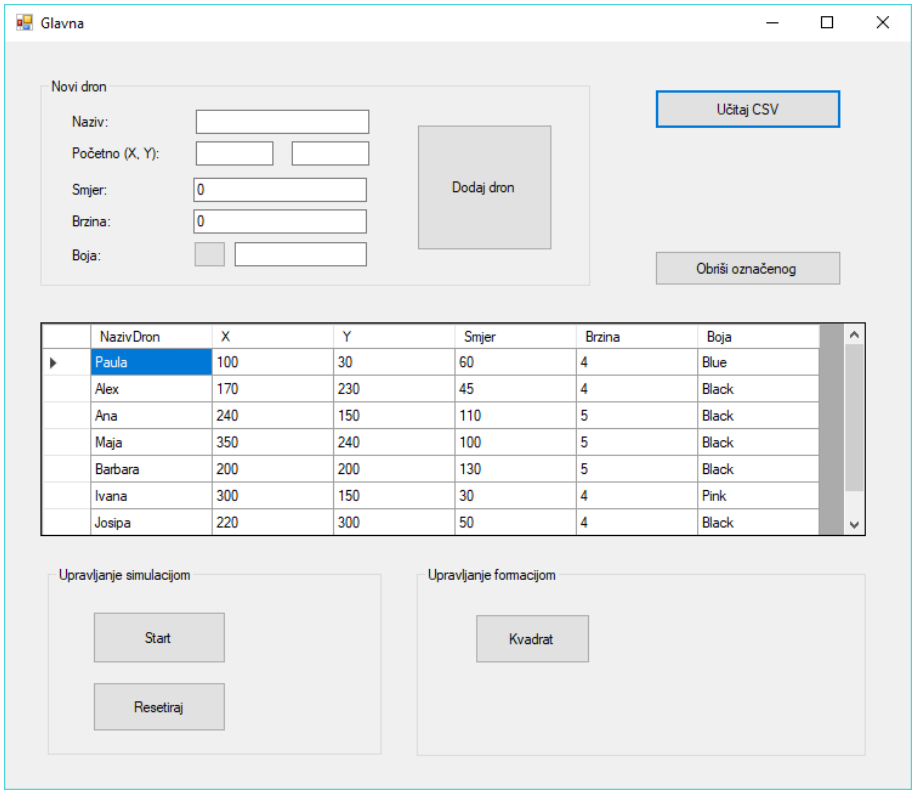
U ovom poglavlju pokazat ću primjer izvođenja aplikacije, tj. simulatora kretanja jata dronova. Kada se aplikacija pokrene, prikazuju se dva prozora: jedan u kojem se unose/učitavaju podaci o dronovima i drugi u kojem se iscrtava/simulira kretanje unesenih dronova. Prikazane prozore možemo vidjeti na slici 6.1.



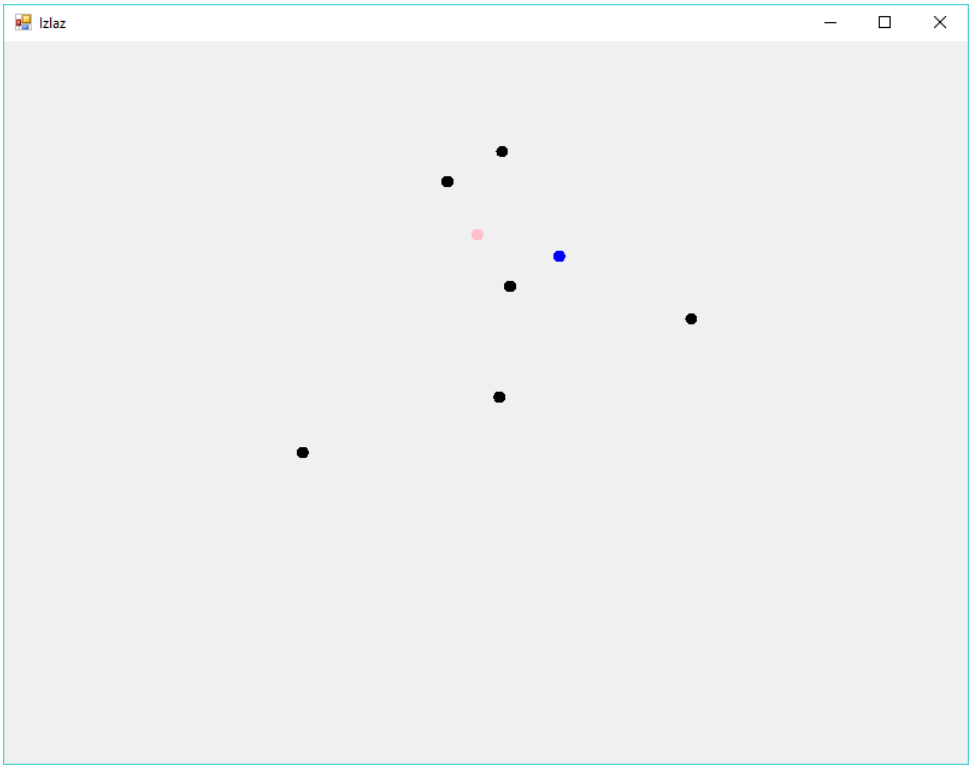
Slika 6.1: Izgled ekrana aplikacije

Osim klasičnog ručnog unosa u polja i dodavanja, postoji mogućnost učitavanja dronova iz csv datoteke. U tom slučaju mora se poštivati zadani format koji jest "ime;x;y;smjer;brzina;boja". Boja se zadaje riječima pa treba biti oprezan. Učitani dronovi vide se u tablici ispod, kao što je prikazano na slici 6.2.

Pritiskom na gumb "Start" započinje simulacija kretanja. Ovisno o postavkama, dronovi će se različito kretati i odbijati od ruba prozora. Klikom na gumb "Kvadrat" počet će samoorganizacija u kvadratni format letenja te će dronovi krenuti prema svojim pozicijama, kao što je vidljivo na slici 6.3.

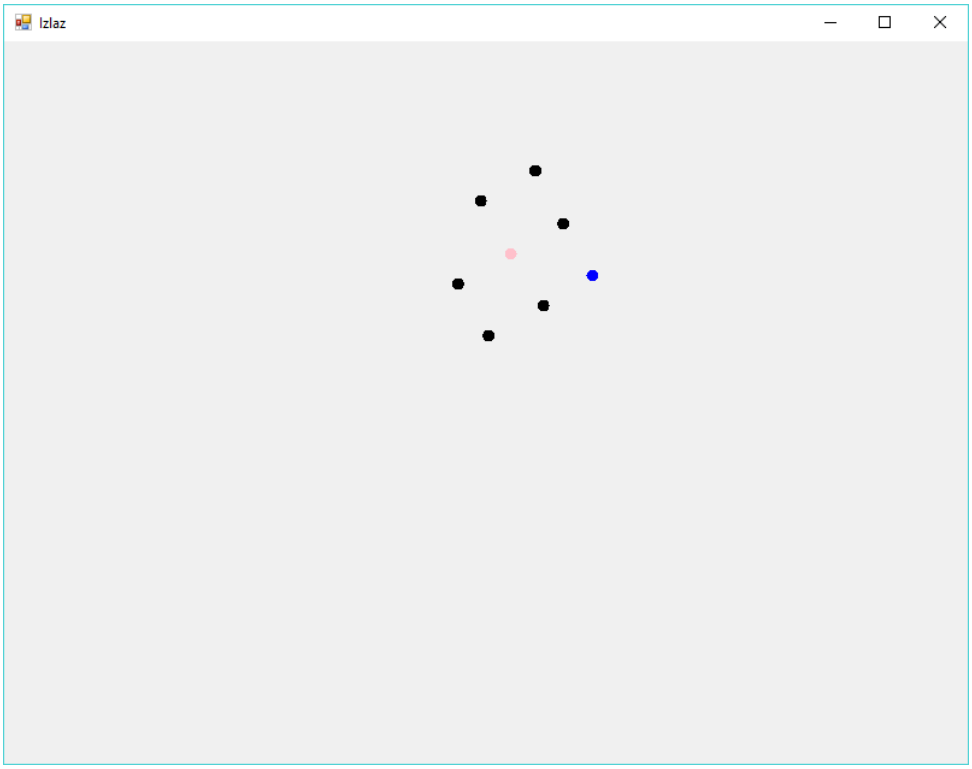


Slika 6.2: Učitani dronovi



Slika 6.3: Kretanje prema svojim pozicijama

U konačnici, kada svi dronovi stignu na svoje pozicije, skupa se u formaciji kreću dalje, kao što se vidi na slici 6.4



Slika 6.4: Kretanje u kvadratnoj formaciji

Poglavlje 7

Zaključak

U ovom radu bavili smo se izradom, opisom i implementacijom višeagentnog sustava unutar jata dronova s ciljem njihove samoorganizacije u zadani format letenja. Budući da je upravljanje dronovima vrlo kompleksno, ovo područje je još uvijek otvoreni problem i dan na razmišljanje, a na dva primjera vidjeli smo i drugačija stajališta i moguća rješenja za postavljene probleme. Ovaj projekt je samo dotakao jednu od mogućnosti moguće komunikacije među agentima u jatu te ima još puno prostora za napredak i dogradnju.

Neke od mogućih dogradnja uključuju nove oblike formacije, kao što su npr. trokutasta ili kružna. Osim toga, potrebno je implementirati ograničenje koje se tiče vidljivosti susjednih dronova. Simulator bi mogao podržavati grupiranje agenata, na način da korisnik označi na karti dronove koje želi grupirati ili formirati u određenu formaciju te da se onda samo ta grupa organizira. Svakako bi trebalo podržati multidretvenost, tako da svakog agenta pokreće vlastita dretva, pa bi i rezultati bili puno realističniji. I naravno, treba težiti decentralizaciji, odnosno pokušati osmisliti algoritam koji bi učinio istu stvar, ali bez vođe i unaprijed generiranih lokacija, već da svi agenti samo prime obavijest o tome koja formacija je odabrana i kolika treba biti njihova međusobna udaljenost, a dalje da uz pomoć lokalnih informacija (od vlastitih susjeda) pokušaju formirati određeni oblik.

Osobno, smatram ovo područje vrlo zanimljivim za daljni rad i istraživanje, a sigurna sam i da može imati veliku primjenu, pogotovo u vojnim svrhama, spasilačkim misijama i sl.

Bibliografija

Agent.GUI, 2018. *Self-organization and Emergence*. Dostupno na `https://www.agentgui.org/index.php/agents-a-mas/self-organization`.

Baker, M., Hexmoor, H. and McLaughlan, B., 2005. *Swarm Control in Unmanned Aerial Vehicles*. In Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI. Las Vegas, Nevada, USA. Dostupno na `https://www.researchgate.net/publication/220834281_Swarm_Control_in_Unmanned_Aerial_Vehicles`.

Jeong, D. and Lee, K., 2014. *Dispersion and Line Formation in Artificial Swarm Intelligence*. Dostupno na `https://www.researchgate.net/publication/263582620_Dispersion_and_Line_Formation_in_Artificial_Swarm_Intelligence`.

TheUAV.com, 2016. *The UAV – Unmanned Aerial Vechile*. Dostupno na `http://www.theuav.com/index.html`.