*Fabijan Lukin, Fran Pregernik, Tomislav Sukser*

# Dokumentacija za izradu programa i senzora za obojivo računarstvo

U Zagrebu, travanj 2006.

***SADRŽAJ***

# 1 Napomena

Ovaj dio dokumentacije pisan je kao dodatak korisničkoj dokumentaciji te bez nje nema previše smisla. Također, u ovom dijelu dokumentacije objašnjene su određene stvari koje nisu detaljno obrađene u tehničkom dijelu dokumentacije. Ovdje obojiva računala promatramo sa strane jednog programera koji piše programe za obojiva računala. Spomenimo još da je platforma potrebna za obogaćivanje ovog projekta ona spomenuta u korisničkoj dokumentaciji i dodatno, Microsoft Visual Studio 2005.

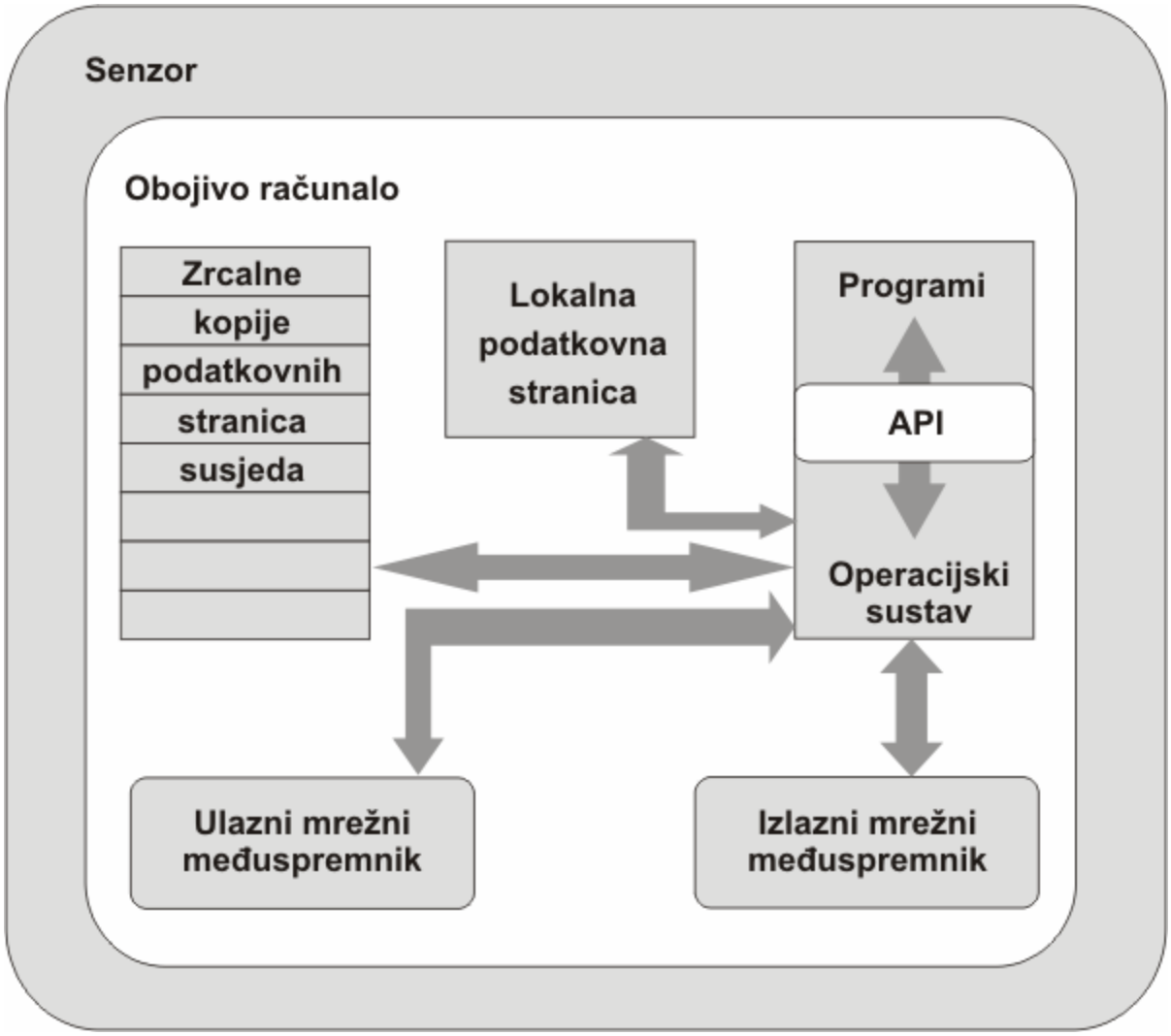
# 2 Dodavanje smisla obojivom računarstvu

Obojiva računala sama za sebe i nisu nešto previše pametno. No dobro, ipak imaju svoj operacijski sustav koji ćemo mi s razine pisca programa promotriti ponajviše sa strane njegovog sučelja – API.

Valjalo bi započeti s programskom reprezentacijom obojivog računala u jezgri simulatora. Za tu svrhu bacit ćemo pogled na razred GenericPaintableComputer koji se nalazi u datoteci PaintableSimulator.Engine\PaintableComputers\GenericPaintableComputer.PaintableComputerBase.cs. Ono što nam je od nekog značaja u ovom trenutku jest jedna metoda razreda - public override void Cycle(). Ona se poziva svako toliko iz jezgre simulatora za svako obojivo računalo. Njezina svrha je da primi sve komunikacijske pakete koji su došli, zatim da pakete koji predstavljaju zrcaljene podatkovne stranice stavi na ispravno mjesto i omogući da programi preko funkcija operacijskog sustava mogu pristupati tim podacima, te da pokrene instalaciju svih primljenih programa. Nakon toga, pokreće svaki program (točnije, njegovu Update metodu), i kada svi programi završe, šalje zrcalnu sliku svoje podatkovne stranice susjedima ukoliko je došlo do promjene na njoj, te šalje programe koji su se naznačili za kopiranje, ako takvih ima. Pseudokod koji bi to najbolje ilustrirao:

```
Cycle()
{
    dok (ima_paketa)
    {
        primi paket p;
        ako je (p == program)
            instaliraj p; // p.Install();
        inače ako je (p == podatkovna stranica)
            osvježi zrcalnu sliku podatkovne stranice p;
    }
    za svaki instalirani program prog
        pokreni prog.Update();
    za svaki program označen za deinstalaciju
        pokreni prog.UnInstall();
    za svako zatraženo kopiranje programa prog
        kopiraj i pošalji paket s programom prog;
```

```
    ako je (podatkovna stranica promijenjena
                ili nije slana već dugo vremena)
        pošalji podatkovnu stranicu;
}
```




*Arhitektura obojivog računala*

## 2.1 Arhitektura programa

U prethodno navedenom pseudokodu spomenute su metode Install, UnInstall i Update od programa. Međutim, to nisu sve metode koje jedan program mora imati. Prikaz metoda i njihovo značenje dan je u sljedećoj tablici.

| Metoda | Objašnjenje |
|---|---|
| void Install(); | Pokreće se prilikom ubacivanja programa u obojivo računalo, program bi u ovoj metodi trebao obaviti sve da može dalje normalno raditi |
| void Uninstall(byte remainingTimeslots); | Pokreće se prije nego što operacijski sustav uništi program i bilokakve njegove tragove. Parametar remainingTimeslots označava vrijeme u taktovima unutar kojeg bi program trebao obaviti ovaj posao. Simulator ne provjerava duljinu trajanja izvođenja. |
| void TransferGranted(ref ProcessFragmentBase copy); | Pokreće se nakon što je odobreno slanje programa dalje. Parametar je referenca na novi program, pri čemu se u ovoj metodi mogu definirati početni parametri programa za računalo na koje se on seli. |
| void TransferRefused(); | Metoda se poziva nakon što je programu odbijen zahtjev za kopiranjem, tj. preseljenjem na drugo računalo. |
| void Update(byte remainingTimeslots); | Ova metoda jest i najvažnija. Ona predstavlja izvođenje programa. Poziva se u svakom ciklusu operacijskog sustava. Parametar remainingTimeslots je broj ciklusa unutar kojeg bi metoda trebala završiti svoje izvođenje, no simulator ne provjerava trajanje izvođenja. |

*Metode programa*

Osim tih metoda program može sadržavati vlastite metode. U svakom slučaju, program koji se piše za simulator mora naslijediti apstraktni razred ProcessFragmentBase koji u sebi sadrži gore navedene apstraktne metode koje se moraju naslijediti i nadjačati (override).

Program koji se piše u naslijeđivanju mora nadjačati i svojstvo public uint ProcessFragmentId koje označava jedinstveni identifikacijski broj programa. To je nužno što program mora posjedovati i kako se on mora napisati da bi se mogao u simulatoru ubaciti u neko obojivo računalo. No, to uistinu nije dovoljno ukoliko želimo prikazati izvođenje tog programa u simulatoru, tj. ukoliko želimo obojati računalo.

## 2.2 Dodavanje boje programu

Nije potrebno puno mudrosti, ali je potrebno napraviti još jedan razred koji nasljeđuje razred HomepageColorizer. Potrebno je nadjačati metodu public abstract Color Colorize(Homepage homepage); koja kao svoj parametar prima podatkovnu stranicu za koju se očekuje da sadrži zapis programa iz kojeg se može odrediti boju kojom će obojivo računalo biti prikazano u simulaciji ako se promatra utjecaj izabranog programa. Metoda Colorize mora vratiti objekt tipa Color koji je točno boja koja će se prikazati u simulatoru. Neka konvencija koja se poštovala u simulatoru jest da je zid crn, računala su u pravilu bijela ili prikazana nekom življom bojom ukoliko se radi o prikazivanju stanja promatranog programa.

## 2.3 Ostale napomene za pisanje programa

Programi za obojiva računala ne smiju iz sigurnosnih razloga baratati nikakvim ulazno-izlaznim operacijama operacijskog sustava na kojem je simulator pokrenut. Ono što oni smiju koristiti jest biblioteka metoda operacijskog sustava obojivog računala i još neke naprednije aritmetičke i slične funkcije. Detalji o pisanju vlastitog programa i spajanju razreda za bojanje s programom bit će dani kasnije u tekstu, poslije opisa biblioteke operacijskog sustava obojivog računala.

# 3 Operacijski sustav obojivog računala i njegova biblioteka

Prije u tekstu vidjeli smo da jedan ciklus izvođenja operacijskog sustava obojivog računala u sebi sadrži razne pozive metoda s kojima on komunicira s programom. No, sada je red objasniti kako program može komunicirati s operacijskim sustavom, jer kao što znamo, on ne može izravno komunicirati niti s drugim programima, niti sa susjednim obojivim računalima i slično. Operacijski sustav omogućava upravo sve te operacije.

U ovom trenutku, podijelit ćemo biblioteke na 5 dijelova, prikazanih u tablici.

| Grupa metoda (biblioteka) | Opis |
|---|---|
| HomePage: Write | Metode za pisanje po podatkovnoj stranici |
| HomePage: Read | Metode za čitanje podataka s podatkovne stranice |
| I/O Space: Read | Metode za čitanje podataka sa zrcaljenih podatkovnih stranica |
| Process Fragment Transfer | Metode za upravljanje prijenosom programa |
| Maintenance & Miscelaneous | Nerazvrstane metode |

*Podjela biblioteka operacijskog sustava*

## 3.1 Biblioteka za pisanje po podatkovnoj stranici

### 3.1.1 Metoda int PostEntry(uint processFragmentId, object post)

Ovo je metoda koja zapisuje objekt post na prvo slobodno mjesto na dijelu podatkovne stranice određenom za program identifikacijskog broja processFragmentId.

| Parametar processFragmentId | Identifikacijski broj programa na čiji dio na stranici se postavlja zapis. |
|---|---|
| Paremetar post | Zapis koji se postavlja na prvo slobodno mjesto. |
| Povratna vrijednost | Redni broj ubačenog zapisa na dijelu podatkovne stranice koji je predviđen za program identifikacijskog broja processFragmentId, odnosno -1 ukoliko zapisivanje nije bilo moguće ostvariti. |

*Prikaz ulaza i izlaza metode PostEntry*

### 3.1.2 Metoda int PostEntry(object post)

Zapisuje objekt post na prvo slobodno mjesto na dijelu podatkovne stranice za program koji je pozvao ovu metodu.

| | |
|---|---|
| Paremetar post | Zapis koji se postavlja na prvo slobodno mjesto. |
| Povratna vrijednost | Redni broj ubačenog zapisa na dijelu podatkovne stranice koji je predviđen za program koji je pozvao ovu metodu, odnosno -1 ukoliko zapisivanje nije bilo moguće ostvariti. |

*Prikaz ulaza i izlaza metode PostEntry*

### 3.1.3 Metoda int PostEntry(uint processFragmentId, int index, object post)

Zapisuje objekt post na mjesto indeksa index u dio podatkovne stranice određen za program identifikacijskog broja processFragmentId.

| | |
|---|---|
| Parametar processFragmentId | Identifikacijski broj programa na čiji dio na stranici se postavlja zapis. |
| Parametar index | Indeks zapisa na dijelu podatkovne stranice na koji će upisivanje biti obavljeno. |
| Paremetar post | Zapis koji se postavlja na mjesto specificirano identifikacijskim brojem programa i indeksom. |
| Povratna vrijednost | Redni broj ubačenog zapisa na dijelu podatkovne stranice koji je predviđen za program identifikacijskog broja processFragmentId, dakle – isti kao i predani parametar index; odnosno -1 ukoliko zapisivanje nije bilo moguće ostvariti. |

*Prikaz ulaza i izlaza metode PostEntry*

### 3.1.4 Metoda int PostEntry(int index, object post)

Zapisuje objekt post na mjesto indeksa index u dio podatkovne stranice određen za program koji je pozvao dotičnu metodu.

| | |
|---|---|
| Parametar index | Indeks zapisa na dijelu podatkovne stranice na koji će upisivanje biti obavljeno. |
| Paremetar post | Zapis koji se postavlja na mjesto specificirano indeksom i identifikacijskim brojem programa koji je pozvao ovu metodu |
| Povratna vrijednost | Redni broj ubačenog zapisa na dijelu podatkovne stranice koji je predviđen za program koji je pozvao metodu, dakle – isti kao i predani parametar index; odnosno -1 ukoliko zapisivanje nije bilo moguće ostvariti. |

*Prikaz ulaza i izlaza metode PostEntry*

### 3.1.5 Metoda bool RemoveEntry(uint processFragmentId, int index)

Briše zapis s dijela podatkovne stranice predviđenog za program identifikacijskog broja processFragmentId na mjestu indeksa index.

| | |
|---|---|
| Parametar processFragmentID | Identifikacijski broj programa na čijem dijelu stranice će se obrisati zapis |
| Paremetar index | Indeks zapisa na dotičnom dijelu stranice koji se briše |
| Povratna vrijednost | true ukoliko je brisanje uspješno izvedeno, false inače |

*Prikaz ulaza i izlaza metode RemoveEntry*

### 3.1.6 Metoda bool RemoveEntry(int index)

Briše zapis s dijela podatkovne stranice predviđenog za program koji je pokrenuo ovu metodu na mjestu indeksa index.

| | |
|---|---|
| Paremetar index | Indeks zapisa na dotičnom dijelu stranice koji se briše |
| Povratna vrijednost | true ukoliko je brisanje uspješno izvedeno, false inače |

*Prikaz ulaza i izlaza metode RemoveEntry*

### 3.1.7 Metoda int RemoveAllEntries(uint processFragmentId)

Briše sve zapise koji pripadaju programu identifikacijskog broja processFragmentId.

| | |
|---|---|
| Paremetar index | Indeks zapisa na dotičnom dijelu stranice koji se briše |
| Povratna vrijednost | Broj obrisanih zapisa. |

*Prikaz ulaza i izlaza metode RemoveAllEntries*

### 3.1.8 Metoda int RemoveAllEntries()

Briše sve zapise koji pripadaju programu koji je pozvao ovu metodu.

| | |
|---|---|
| Povratna vrijednost | Broj obrisanih zapisa. |

*Prikaz ulaza i izlaza metode RemoveAllEntries*

## 3.2 Biblioteka za čitanje podataka s podatkovne stranice

### 3.2.1 Metoda int GetPostCount(uint processFragmentId)

Metoda GetPostCount dohvaća broj zapisa koji su zapisani na dijelu podatkovne stranice predviđenom za program identifikacijskog broja processFragmentId.

| | |
|---|---|
| Paremetar processFragmentId | Identifikacijski broj programa za koji će se pregledati broj zapisa na podatkovnoj stranici |
| Povratna vrijednost | Broj zapisa na dotičnom dijelu podatkovne stranice |

*Prikaz ulaza i izlaza metode GetPostCount*

### 3.2.2 Metoda int GetPostCount()

Ova metoda GetPostCount dohvaća broj zapisa koji su zapisani na dijelu podatkovne stranice predviđenom za program koji je pozvao ovu metodu.

| | |
|---|---|
| Povratna vrijednost | Broj zapisa na dijelu podatkovne stranice predviđenom za program koji je pozvao ovu metodu |

*Prikaz ulaza i izlaza metode GetPostCount*

### 3.2.3 Metoda object GetPost(uint processFragmentId, int index)

Ova metoda dohvaća zapis na mjestu indeksa index iz dijela podatkovne stranice predviđene za program identifikacijskog broja processFragmentId.

| | |
|---|---|
| Paremetar processFragmentId | Identifikacijski broj programa s čijeg dijela podatkovne stranice će se zapis dohvatiti. |
| Parametar index | Indeks zapisa na dotičnom dijelu podatkovne stranice. |
| Povratna vrijednost | Zapis koji se nalazi na traženom mjestu, null inače. |

*Prikaz ulaza i izlaza metode GetPost*

### 3.2.4 Metoda GetPost(int index)

Ova metoda dohvaća zapis na mjestu indeksa index iz dijela podatkovne stranice predviđene za program koji je pozvao ovu metodu.

| | |
|---|---|
| Parametar index | Indeks zapisa na dijelu podatkovne stranice predviđenom za program koji je pozvao ovu metodu. |
| Povratna vrijednost | Zapis koji se nalazi na traženom mjestu, null inače. |

*Prikaz ulaza i izlaza metode GetPost*

## 3.3 Biblioteka za čitanje podataka sa zrcaljenih podatkovnih stranica

### 3.3.1 Metoda int GetIOSpaceCount()

Metoda dohvaća broj zrcaljenih podatkovnih stranica.

| | |
|---|---|
| Povratna vrijednost | Broj zrcaljenih podatkovnih stranica |

*Prikaz ulaza i izlaza metode GetIOSpaceCount*

### 3.3.2 Metoda int GetIOPostCount(int mirrorIndex, uint processFragmentId)

Metoda dohvaća broj zapisa na zrcaljenoj podatkovnoj stranici rednog broja susjeda mirrorIndex, za dio stranice koji pripada programu processFragmentId.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda čija se podatkovna stranica promatra |
| Parametar processFragmentId | Identifikacijski broj programa čiji se dio dotične zrcaljene stranice pregledava. |
| Povratna vrijednost | Broj zapisa koji se nalaze na dotičnom dijelu zrcaljene podatkovne stranice. |

*Prikaz ulaza i izlaza metode GetIOSpaceCount*

### 3.3.3 Metoda int GetIOPostCount(int mirrorIndex)

Metoda dohvaća broj zapisa na zrcaljenoj podatkovnoj stranici rednog broja susjeda mirrorIndex, za dio stranice koji pripada programu istog identifikacijskog broja kao i program koji je pozvao ovu metodu.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda čija se podatkovna stranica promatra |
| Povratna vrijednost | Broj zapisa koji se nalaze na dotičnom dijelu zrcaljene podatkovne stranice. |

*Prikaz ulaza i izlaza metode GetIOSpaceCount*

### 3.3.4 Metoda object GetIOPost(int mirrorIndex, uint processFragmentId, int postIndex)

Dohvaća zapis koji se nalazi na dijelu zrcaljene podatkovne stranice, rednog broja susjeda mirrorIndex, koji je predviđen za program identifikacijskog broja processFragmentId, na mjestu indeksa postIndex.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda čija se podatkovna stranica promatra |
| Parametar processFragmentId | Identifikacijski broj programa čiji se dio dotične zrcaljene stranice pregledava. |
| Parametar postIndex | Indeks zapisa na dotičnom dijelu zrcaljene stranice. |
| Povratna vrijednost | Zapis koji se nalazi na tom mjestu, null inače. |

*Prikaz ulaza i izlaza metode GetIOPost*

### 3.3.5 Metoda object GetIOPost(int mirrorIndex, int postIndex)

Dohvaća zapis koji se nalazi na dijelu zrcaljene podatkovne stranice, rednog broja susjeda mirrorIndex, koji je predviđen za program identifikacijskog broja istog identifikacijskom broju programa koji je pozvao ovu metodu, na mjestu indeksa postIndex.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda čija se podatkovna stranica promatra |
| Parametar postIndex | Indeks zapisa na dijelu dotične zrcaljene stranice, koji je predviđen za program koji ima isti identifikacijski broj kao i pozivajući program. |
| Povratna vrijednost | Zapis koji se nalazi na tom mjestu, null inače. |

*Prikaz ulaza i izlaza metode GetIOPost*

## 3.4 Biblioteka za upravljenje prijenosom programa

### 3.4.1 Metoda bool QueueTransfer(int mirrorIndex)

Ovo je metoda koja stavlja program koji poziva ovu metodu u red čekanja za slanje na susjeda rednog broja mirrorIndex. Pretpostavljena vrijednost prioriteta za slanje je 0.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda na koji se želi poslati program. |
| Povratna vrijednost | true ako je ubacivanje u red za slanje uspjelo, false inače |

*Prikaz ulaza i izlaza metode QueueTransfer*

### 3.4.2 Metoda bool QueueTransfer(int mirrorIndex, byte priority)

Ovo je metoda koja stavlja program koji poziva ovu metodu u red čekanja za slanje na susjeda rednog broja mirrorIndex.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda na koji se želi poslati program. |
| Parametar byte | 8-bitni broj koji predstavlja prioritet slanja |
| Povratna vrijednost | true ako je ubacivanje u red za slanje uspjelo, false inače |

*Prikaz ulaza i izlaza metode QueueTransfer*

### 3.4.3 Metoda bool IsQueuedForTransfer()

Provjerava jesu li svi zahtjevi za slanje posluženi.

| | |
|---|---|
| Povratna vrijednost | true ako ima neposluženih zahtjeva za slanje, false inače |

*Prikaz ulaza i izlaza metode IsQueuedForTransfer*

### 3.4.4 Metoda bool IsQueuedForTransfer(int mirrorIndex)

Provjerava jesu li svi zahtjevi za slanje na susjeda rednog broja mirrorIndex posluženi.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda za kojega se traži provjera posluženih zahtjeva. |
| Povratna vrijednost | true ako ima neposluženih zahtjeva za slanje na susjeda rednog broja mirrorIndex, false inače |

*Prikaz ulaza i izlaza metode IsQueuedForTransfer*

### 3.4.5 Metoda bool ClearQueueTransfer(int mirrorIndex)

Poništava slanje programa koji je pozvao ovu metodu, tj. briše ga iz reda čekanja.

| | |
|---|---|
| Parametar mirrorIndex | Redni broj susjeda za kojega se traži poništenje slanja programa |
| Povratna vrijednost | true ako je poništenje slanja uspjelo, false inače |

*Prikaz ulaza i izlaza metode ClearQueueTransfer*

### 3.4.6 Metoda bool QueueUninstall()

Stavlja program koji je pozvao ovu metodu u red za deinstalaciju s dotičnog računala. Pretpostavljeni prioritet deinstalacije je 0.

| | |
|---|---|
| Povratna vrijednost | true ako je stavljanje u red za deinstalaciju uspjelo, false inače |

*Prikaz ulaza i izlaza metode QueueUninstall*

### 3.4.7 Metoda bool QueueUninstall(byte priority)

Stavlja program koji je pozvao ovu metodu u red za deinstalaciju s dotičnog računala, pri čemu je priority predstavlja prioritet.

| | |
|---|---|
| Parametar priority | 8-bitni broj koji predstavlja prioritet deinstalacije |
| Povratna vrijednost | true ako je stavljanje u red za deinstalaciju uspjelo, false inače |

*Prikaz ulaza i izlaza metode QueueUninstall*

## 3.5 Biblioteka sa nerazvrstanim metodama

### 3.5.1 Metoda uint GetComputerId()

Ova metoda vraća identifikacijski broj obojivog računala na kojem se program, koji ju je pozvao, nalazi.

| | |
|---|---|
| Povratna vrijednost | Identifikacijski broj računala na kojem je program pokrenut |

*Prikaz ulaza i izlaza metode GetComputerId*

# 4 Senzori

Senzori su nadogradnja obojivih računala. Kao takva posjeduju sve njihove metode i funkcionalnosti, a dodaju i nekoliko novih.

## 4.1 Sučelje ISensor

ISensor jest sučelje senzora. Drugim riječima, to su jedine metode koje morate imati u senzoru kako bi on mogao raditi u simulatoru. Sve ostale koje napišete možete koristiti samo vi.

Radi se o slijedećim metodama:

### 4.1.1 Type[] DataFeeders

Jedan senzor može primati od simulatora više različitih podataka, npr. temperaturu i vlažnost, i na osnovu oba podatka prikazivati se drukčije na ekranu. S ovom metodom kažete simulatoru s kojim tokovima podataka senzor radi.

| | |
|---|---|
| Povratna vrijednost | Polje tipova toka podataka |

*Opis metode Type[] DataFeeders*

### 4.1.2 void SetEnvironmentValue(Type dataFeederType, double sensorValue)

Na osnovu prethode funkcije, simulator povremeno dostavlja podatke senzoru preko ove metode.

| | |
|---|---|
| Parametar dataFeederType | Tip dobavljača podataka |
| Parametar sensorValue | Vrijednost podatka koju je vratio kod podataka |

*Opis metode SetEnvironmentValue*

### 4.1.3 HomepageColorizer SensorColorizer

Da bi se nešto prikazalo na ekranu, senzor treba imati i svoju funkciju za bojanje. Ovom metodom kažete simulatoru koja je to funkcija.

| | |
|---|---|
| Povratna vrijednost | Objekt tip HomepageColorizer koji obavlja bojanje |

*Opis metode SensorColorizer*

## 4.2 Promjena obojivog računala

Da bi sve radilo kako treba moramo promijeniti samo jednu metodu GenericPaintableComputer-a, razreda kojeg nasljeđujemo. Unutar te metode podatke na osnovu kojih vršimo bojanje moramo upisati u podatkovnu stranicu (HomePage), jer funkcije za bojenje samo tome mogu pristupati.

### 4.2.1 public override void Cycle()

Unutar nje, nakon upisivanje vrijednosti senzora u lokalnu memoriju, moramo pozvati i osnovnu metodu Cycle().

## 4.3 PressureSensorComputer

Najbolje da napišemo jedan senzor. Stvorite novu datoteku PressureSensorComputer.cs u projektu PaintableSimulator.Engine, u podmapi „PaintableComputers\Sensors“. Početi ćemo od kostura programa u kojemu samo kažemo ime senzora, da nasljeđujemo GenericPaintableComputer i da imamo senzore u njemu.

```
public class PressureSensorComputer : GenericPaintableComputer,
ISensor

{

    public const uint ProcessFragmentId = 0x12121212;

}
```

ProcessFragmentId mora biti jednistven, bez obzira radi li se o obojivom računalu ili senzoru.

To nam je bitno prilikom pristupanja lokalnoj memoriji, jer tako možemo pristupati memoriji pojedinog programa ili senzora. To koristimo u konstruktoru senzora.

```
public PressureSensorComputer(uint maximumNeighbourCount)

    : base(maximumNeighbourCount)

{

    localHomepage.Add(ProcessFragmentId,                    new
HomepagePostCollection());

}
```

Ovaj konstruktor je generički, i nije ga potrebno niti preporučeno mijenjati.

Simulatoru moramo reći kako ćemo bojati sebe. To se radi na isti način kao i za programe u poglavlju 5.4 i to uz pomoć DefaultHomepageColorizer atributa kojem je parametar tip razreda objekta za bojanje.

```
[DefaultHomepageColorizer(typeof(PressureSensorColorizer))]

public class PressureSensorComputer : GenericPaintableComputer,
ISensor

{ … }
```

Da bi dobili podatke o okolini, moramo reći simulatoru što želimo. To radimo u metodi DataFeeders, gdje vraćamo popis svih dobavljača podataka koje znamo koristiti. Mi koristimo AntWalkerDataFeeder, koji simulira kruženje mrava oko središta simulatora. Dobavljači moraju postojati kako bi ih mogli koristiti. Ova se metoda poziva samo jednom, pa je sigurno svaki puta stvarati novi dobavljač. Drugi način je preko privatne varijable, kao u primjeru iznad.

```
public Type[] DataFeeders

{

    get

    {

        return new Type[] { typeof(AntWalkerDataFeeder) };

    }

}
```

Naravno, povremeno će simulator nama slati podatke od dobavljača podataka. U ovom slučaju, AntWalkerDataFeeder-a. Mi te podatke primamo kroz metodu SetEnviromentValue. Jedan parametar je vrijednost, a drugi je tip dobavljača podataka koji je taj podatak postalo. U ovoj funkciji je bitno provjeriti o kojem se dobavljaču radi, da, ako ih primamo više, znam što ćemo s njim napraviti. Ili, ako ne znamo što bi s njim, da ga možemo ignorirati. Vrijednosti je potrebno spremiti u privatne varijable, kako bi ih poslije mogli koristiti.

```
private double pressure;

public void SetEnvironmentValue(Type dataFeederType, double
sensorValue)

{

    if (dataFeederType == typeof(AntWalkerDataFeeder))

    {

        pressure = sensorValue;

    }

}
```

Do sada smo pokrili dio koji je striktno vezan za razred ISimulator. Još moramo prekriti Cycle() metodu razreda GenericPaitableComputer. U njoj sve podtake koje smo dobili od dobavljača podataka i spremili u privatne varijable, sada upisujemo u lokalnu memoriju u obliku koji objekt za bojanje zna prepoznati.

```
public override void Cycle()

{

    UpdateSensorHomepage();

    base.Cycle();

}

private void UpdateSensorHomepage()

{

    base.PostEntry(ProcessFragmentId, 0, pressure);

}
```

Metoda UpdateSensorHomepage nije komplicirana pa se je mogla i cijela smjestiti unutar Cycle metode, ali preporučujemo ovako programirati jer je preglednije. Na kraju naše Cycle metode, obvezno pozvati baznu metodu!

## 4.4 Kako obojati senzor

Za to moramo napraviti razred koja obavlja bojanje. Za to ćemo napisati novi razred koji će naslijediti razred HomepageColorizer, i nadjačati metodu Colorize. U njoj računamo boju kojom će simulator prikazati senzor. To radimo tako da pretražujemo imamo li u lokalnoj memoriji zapis našeg senzora. Ako imamo, pročitajmo ga i pretvorimo u boju. U ovom slučaju, nijansu žute.

```
public class PressureSensorColorizer : HomepageColorizer

{

    public override Color Colorize(Homepage homepage)

    {

        Color retval = Color.WhiteSmoke;

        for (int i = 0; i < homepage.Count; i++)

            if (homepage.KeyList[i] ==
PressureSensorComputer.ProcessFragmentId)

            {

                double pressure = 0.0;
```

```
                    if (homepage.DataList[i].Count == 0)
                        break;
                    pressure = (double)homepage.DataList[i][0];
                    int color = (int)(255 * pressure / 10.0);
                    if (color > 255)
                        color = 255;
                    retval = Color.FromArgb(color, color, 0);
                    break;
                }
                return retval;
            }
        }
```

## 4.5 Naš mrav hoda!

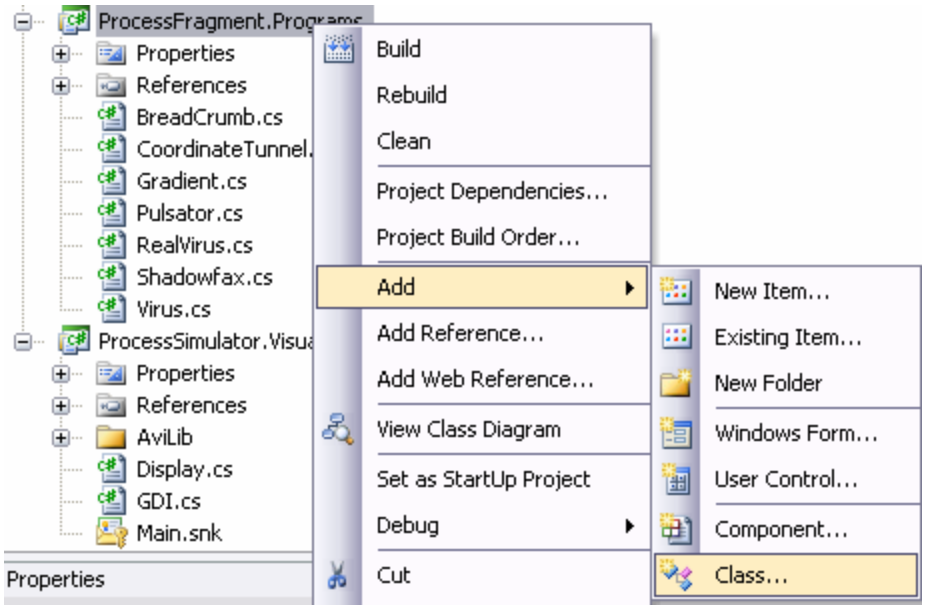
I to je to! Ako je dobavljač vrijednosti napisan dobro, naš senzor će prikazivati hodanje mrava u krug.

Za stvarnu implementaciju senzora i razreda za bojanje, pogledajte datoteku TemperatureSensorPaintableComputer.cs, a za dobavljač podataka datoteku AntWalkerDataFeeder.cs.

# 5 Pisanje svog programa

## 5.1 Dodavanje razreda

Nakon svega ovog, bio bi red da pokažemo pisanje jednog jednostavnog programa. Dakle, potrebno je otvoriti projekt ProcessFragment.Programs, i napraviti novi razred u njemu. Mi ćemo napraviti novi razred imena mojVirus i spremiti ga u datoteku MojVirus.cs.




*Dodavanje novog razreda*

Sada je stvoren novi razred MojVirus unutar imenika (namespace) PaintableSimulator.Programs. Razred kao takav nije dovoljan da bi on postao program. On mora naslijediti apstraktni razred ProcessFragmentBase. Prije toga, potrebno je uključiti navedene using direktive:

```
using PaintableSimulator.Engine;

using PaintableSimulator.Engine.PaintableComputers.OperatingSystem;

using PaintableSimulator.Engine.PaintableComputers;

using System.Drawing;
```

Nakon toga trebamo naslijediti razred MojVirus iz razreda ProcessFragmentBase, naredbom iz izbornika Implement Abstract Class. Sada razred sadrži jedno svojstvo – ProcessFragmentId i pet metoda, redom: Install, Uninstall, TransferGranted, TransferRefused, te na kraju Update.

Za početak ćemo implementirati svojstvo ProcessFragmentId. Ono mora vratiti 32-bitni nepredznačeni cijeli broj, koji će biti jedinstven u svim programima. Dobar izbor bi mogao biti korištenje JMBAG-a, no mi ćemo za ovaj primjer odabrati broj 0x1369. I to ćemo

napisati. Dakle, svojstvo vraća 0x1369, a kako je moguće samo njegovo čitanje, dio za pisanje nećemo implementirati; ipak ne bi bilo previše zgodno kada bi program mijenjao svoj identifikacijski broj tijekom izvođenja.

```
public override uint ProcessFragmentId

{

    get { return 0x1369; }

}
```

Zatim, ukratko ćemo proći kroz metode Uninstall, TransferGranted i TransferRefused. Ovaj naš virus će imati svojstvo da se širi na sve susjede na kojima se još ne nalazi i da stalno provjerava jesi li svi njegovi susjedi zaraženi, stoga nas uopće ne zanima metoda TransferRefused. Svejedno ćemo kasnije probat ponovo. Dakle, ta metoda ostaje prazna i ne radi ništa. Kad program dobije dopuštenje da kopiranje na neko drugo računalo, pozvat će se metoda TransferGranted. Trebamo li kopiranome programu dati neku informaciju koju će on vidjeti kada dođe na novo obojivo računalo, napisat ćemo nešto u tijelu metode TransferGranted. Kako to kod nas nije slučaj jer se od programa traži uvijek jednako ponašanje, i ova metoda će nam ostati prazna. Nadalje, u metodi Uninstall bi trebali napisati kod koji će počistiti sve što naš program ostavi u obojivom računalu. Postoji razlog zašto ćemo i ovu metodu ostavit praznom. Razlog se svodi na to da mi ustvari ne želimo deinstalirati naš virus i obrisat njegove tragove. U protivnom bi mu valjalo promjeniti ime u nešto benignije (iako i ovako nije pretjerano štetan). Na kraju, ove 3 metode u kodu izgledaju ovako:

```
public override void Uninstall(byte remainingTimeslots)

{

    // Ništa.

}


public override void TransferGranted(ref ProcessFragmentBase copy)

{

    // Ništa.

}


public override void TransferRefused()

{

    // Ništa.

}
```

## 5.2 Dodavanje smisla postojanju

Nakon rješavanja dosadnog dijela, rješit ćemo i zabavniji. U tom smislu, napisat ćemo tijelo metoda Install i Update. Za početak, razložit ćemo djelovanje virusa na 2 dijela. Jedno će biti njegov dolazak na novo računalo, a drugi dio će biti njegovo širenje. Prvi dio upravo odgovara metodi Install. Kako nam je dovoljan jedan virus po računalu, njegovu instalaciju ćemo dozvoliti samo onda ukoliko se on već ne nalazi na tom računalu. To ćemo provjeriti na jednostavan način da pogledamo koliko ima zapisa za program ovog identifikacijskog broja na podatkovnoj stranici. Ako ih ima, odmah ćemo deinstalirati novi program i ostaviti stari da živi. U protivnom, zapisat ćemo na naše mjesto na podatkovnoj stranici da smo se doselili. Prema tome, tijelo funkcije Install bi uz naše želje trebalo izgledati ovako:

```
public override void Install()
{
    // Ako postoje naznake da je virus već ovdje...
    if (Api.GetPostCount() > 0)
    {
        // ...deinstalirat ćemo ga...
        Api.QueueUninstall();
        return;
    }
    else
    {
        // ...a u protivnom postavit ćemo na zapis nultog indeksa
        // na stranicu predviđenu za ovaj program oznaku da smo
        // ovdje.
        Api.PostEntry(0, "MojVirus");
    }
}
```

Nakon što smo rješili postavljanje virusa, trebamo rješiti i njegovo širenje. U tom dijelu nema previše mudrosti. Provjerit ćemo zrcaljene stranice svih susjeda od našeg računala i na svako računalo koje na svojoj stranici nema ćemo poslati njegovu kopiju. Kako bi osigurali konstantnu provjeru da na našem zidu ne bi postojalo niti jedno računalo bez virusa, opisani postupak ćemo smjestiti u metodu Update koja će se pozivati nakon promjena na lokalnoj podatkovnoj stranici ili pri uočenoj promjeni na nekoj od zrcaljenih podatkovnih stranica. U konačnici, tijelo metode Update će izgledati ovako:

```
public override void Update(byte remainingTimeslots)
{
    // Ići ćemo iteratorom i po svim susjedima
    for (int i = 0; i < Api.GetIOSpaceCount(); i++)
    {
        // i ako dođemo do nekog (i-tog) koji nema oznaku
        // da je zaražen virusom, zahtijevat ćemo
        // kopiranje na njega (na i-tog susjeda).
        if (Api.GetIOPostCount(i) <= 0)
        {
            Api.QueueTransfer(i);
        }
    }
}
```

## 5.3 Dodavanje boje

Do sada smo napravili jedan vrlo jednostavan program. No, rezultat njegovog izvođenja nećemo vidjeti na lijepi način, odnosno, jedini način na koji možemo vidjeti njegovu prisutnost je taj da iskoristimo opciju Debug u simulatoru i pregledamo sadržaj podatkovnih stranica. To nam ne odgovara pa ćemo u tu svrhu napisati jedan razred koji će poslužiti za vizualizaciju programa. Razred koji napravimo mora biti naslijeđen iz razreda HomepageColorizer i potrebno je nadjačati metodu Colorize. Deklarirat ćemo naš razred imenom BojanjeVirusa i naslijediti ga iz razreda HomepageColorizer. Nakon toga, u razredu se pojavljuje metoda public override System.Drawing.Color Colorize(Homepage homepage). Njezin je zadatak da iz zapisa s podatkovne stranice vrati boju s kojom će se pokazati obojivo računalo na zidu. Možemo reći da ćemo nezaražena računala obojati u zeleno, a zaražena u crveno. Ako naiđemo na zapis od programa MojVirus na podatkovnoj stranici, smatrat ćemo računalo zaraženim, nezaraženim inače.

Podatkovna stranica sadrži popis ključeva gdje oni imaju vrijednosti identifikacijskih brojeva programa koji su zapisali nešto na podatkovnu stranicu. Proći ćemo kroz sve ključeve i za ključ koji predstavlja identifikacijski broj našeg programa, ukoliko postoji bilokoji zapis, vratit ćemo crvenu boju iz metode. Na kraju, ako nismo pronašli odgovarajući ključ, računalo smatramo nezaraženim i predstavljamo ga zelenom bojom. Kod navedene metode u razredu BojanjeVirusa će na kraju izgledati ovako:

```
public override System.Drawing.Color Colorize(Homepage homepage)
{
   // Prolazimo kroz sve zapise na podatkovnoj stranici s
    // iteratorom i.
    for (int i = 0; i < homepage.Count; i++)
    {
        // Ukoliko nađemo zapis koji odgovara programu koji ima
        // identifikator kao i MojVirus (0x1369), ...
        if (homepage.KeyList[i] == 0x1369)
        {
            // ... te ako na tom dijelu podatkovne stranice postoje
zapisi ...
            if (homepage.DataList[i].Count > 0)
            {
                // ... obojat ćemo računalo u crveno.
                return Color.Red;
            }
        }
    }

    // U protivnom, obojat ćemo računalo u zeleno.
    return Color.Green;
}
```

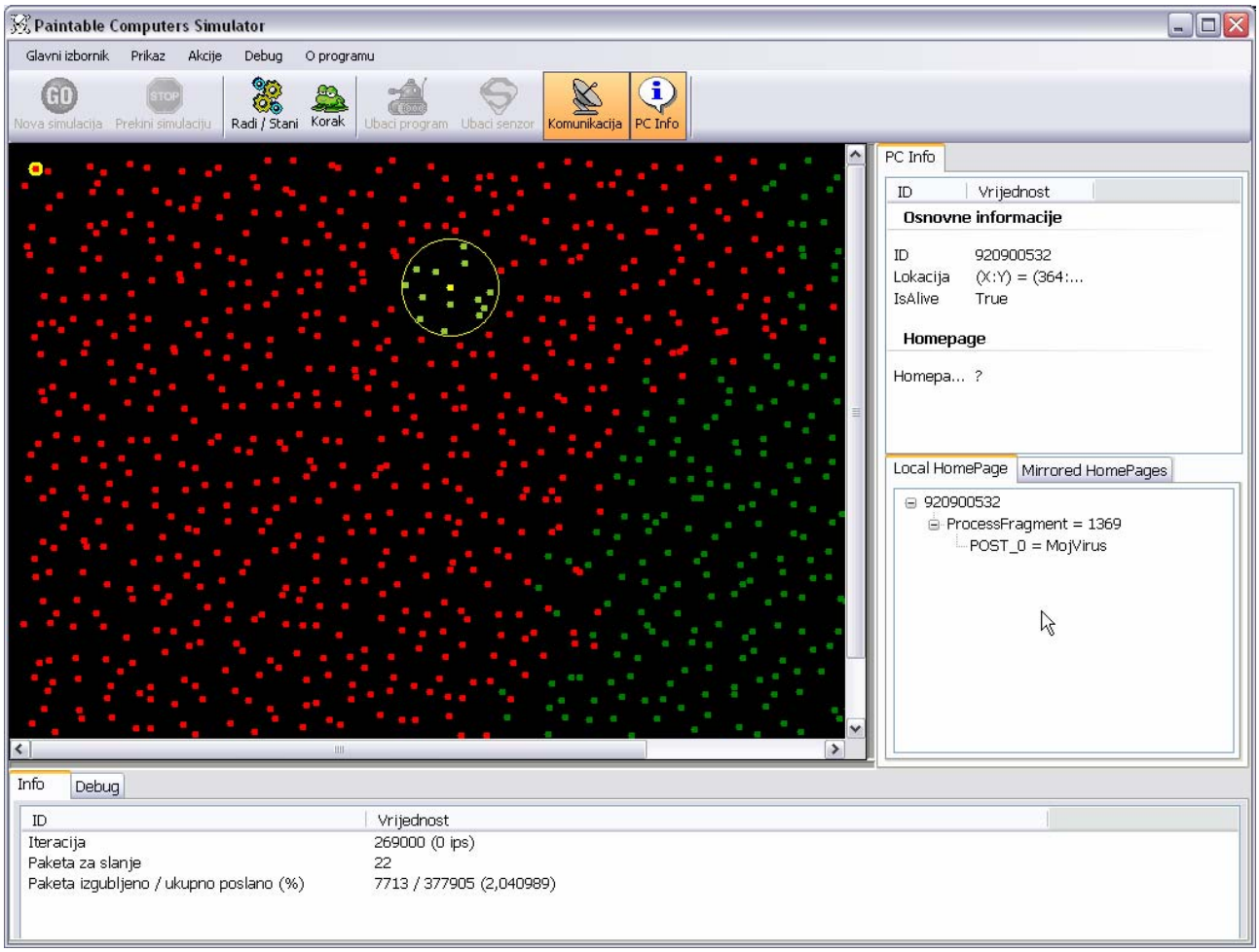
## 5.4 Završne pripreme

Na kraju ostaje nam povezati program i razred koji je odgovoran za bojanje. Za ostvarenje toga, razredu MojVirus pridjelit ćemo nekoliko atributa. Prvi od njih je DefaultHomepageColorizerAttribute, čiji parametar je tip razreda BojanjeVirusa. Sljedeća dva atributa su FriendlyDescription i FriendlyName, a oni predstavljaju opis programa i njegov naziv, respektivno. Njihovi su parametri varijable tipa string. Na kraju, dodat ćemo i atribut Serializable. Nakon ovoga, iznad razreda će se pojaviti sljedeći programski kod:

```
[DefaultHomepageColorizerAttribute(typeof(BojanjeVirusa))]

[FriendlyDescription("Ovo je svježe napisani program")]

[FriendlyName("MojVirus")]

[Serializable()]

public sealed class MojVirus : ProcessFragmentBase { ... }
```

## 5.5 Novi program na djelu

Osim navedenih dodataka, potrebno je oba razreda proglasiti javnima i zapečaćenima, dakle napisati public sealed ispred svakog od njih. Na kraju, preostaje nam da prevedemo naš program i pokrenemo ga. Prevođenje napisanih programa se po postavkama projekta obavlja u DLL datoteku PaintableSimulator.Programs.dll u podmapi Plugins, a dotična podmapa se nalazi u mapi koja je u projektu označena kao ciljna mapa za projekt. Tu datoteku trebamo smjestiti u odgovarajuću mapu koja se nalazi gdje i simulator. Na kraju, možemo pokrenuti simulator i diviti se remek-djelu.




*MojVirus nakon nekoliko koraka simulacije*

# 6 Sigurnost programa

O sigurnosti je već ponešto rečeno u korisničkoj dokumentaciji, no sada ćemo utvrditi gradivo. Programi koji se mogu ubaciti u simulator ne smiju koristiti nikakve ulazno-izlazne operacije niti bilo kakve druge operacije koje bi mogle naškoditi operacijskom sustavu koji je ugostio simulator. Kao primjer toga, među programima se nalazi program RealVirus koji bi trebao obrisati mapu c:\test. Isječak koda iz njegove update procedure:

```
public override void Update(byte remainingTimeslots)

{

    ...

    Directory.Delete("c:\\test");

    ...

}
```

No, zbog ove naredbe simulator neće dozvoliti izvođenje programa RealVirus, te će ga obrisati iz računala u kojem se nalazi. Stoga, kod pisanja vlastitih programa treba pripaziti da se ne koriste nedopuštene biblioteke. Uostalom, za kompleksne programe poput Shadowfaxa nije trebalo ništa više od metoda operacijskog sustava obojivog računala i imenika System.Collections.

# 7 Primjeri

U sljedećem dijelu ovog dokumenta dat ćemo kratak pregled napisanih programa i objasniti njihov rad, s pokojim retkom pseudokoda ili nekom jednostavnom ilustracijom. Detaljnije o svakom programu može se saznati iz izvornog koda koji je komentiran.

## 7.1 Objašnjen rad programa Gradient

Program Gradient predstavlja gradijent na zidu, čija vrijednost ovisi o udaljenosti od izvora, pri čemu se izvorom smatra računalo u koje je inicijalno ubačen program Gradient. Udaljenost se mjeri u skokovima (engl. hops) koji predstavljaju broj potrebnih zrcaljenja da bi podatak s izvora došao do računala za koje se računa gradijent. Prilikom inicijalnog učitavanja programa Gradient u portal, vrijednost skokova je postavljena na nulu, dok prilikom instalacije programa u bilo koje drugo računalo, taj podatak dolazi uvećan za jedan. Ovim postupkom bi načelno svaki program trebao vidjeti susjede kojima je vrijednost skokova manja za 1 od njega samog. No, ukoliko to nije tako i postoje susjedi koji imaju još manju vrijednost broja skokova, program se postavlja na vrijednost tog broja uvećanu za 1 jer postoji način da u tolikom broju skokova podatak zrcaljenjem doputuje od izvora do njega. Ovdje je dan pseudokod metode Update programa Gradient:

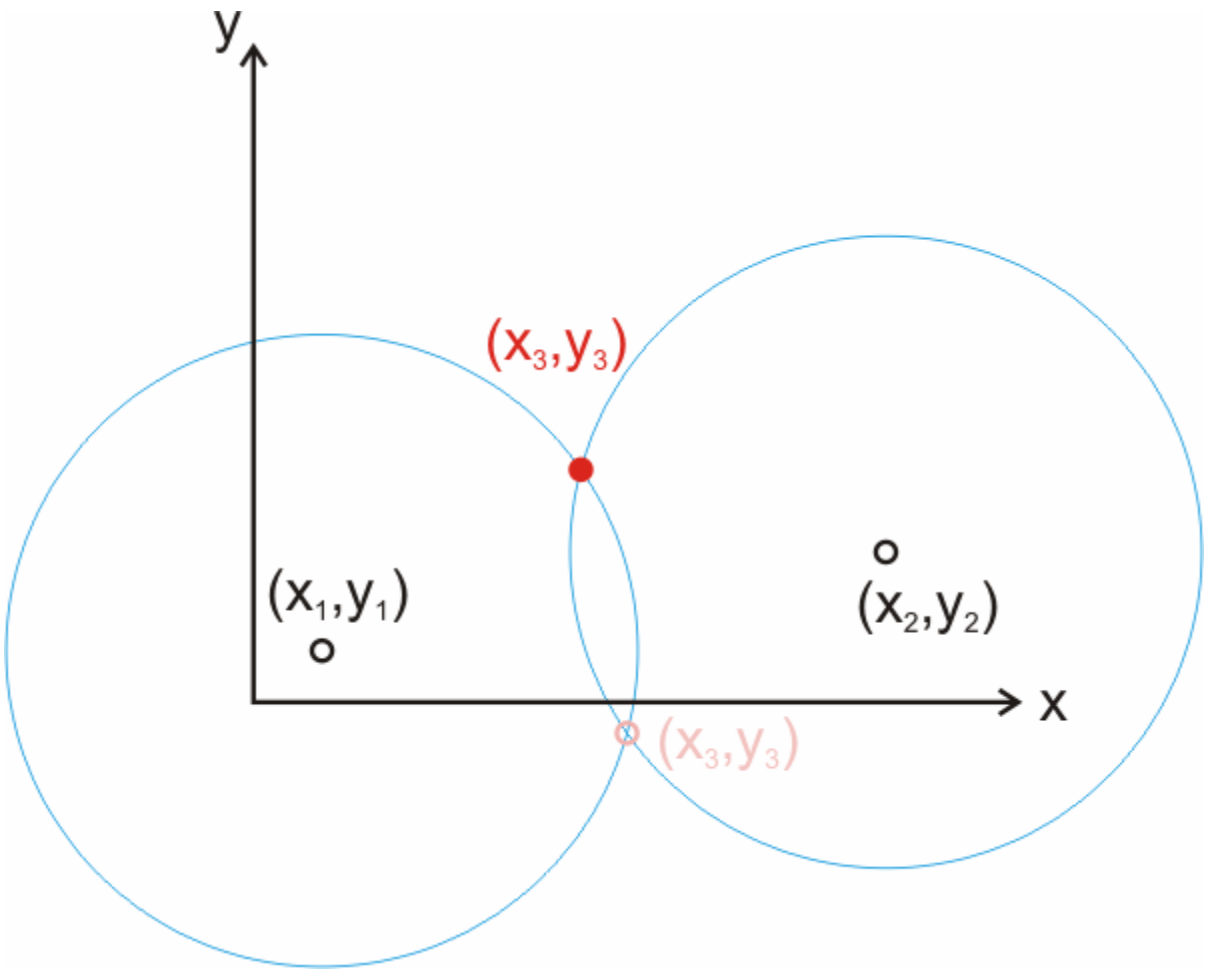
```
// brojSkokova je zapisan kao podatak na podatkovnoj stranici

Update()

{

    prođi kroz sve susjede S

    {

        ako (S nema Gradient)

        {

            kopiraj Gradient na S;

        }

        inače ako (S.brojSkokova < brojSkokova – 1)

        {

            brojSkokova = S.brojSkokova + 1;

        }

    }

}
```

## 7.2 Objašnjen rad programa Coordinate Tunnel

Ovaj program sastoji se od tri dijela nazvana maštovitim nazivima First, Second i Third. Prvi dio nije ništa pametnije od običnog gradijenta. Drugi program je gradijent koji zna izračunati koliko jedan skok između podatkovnih stranica u prosjeku vrijedi za fiktivnu jediničnu udaljenost u koordinatnom sustavu (nazvat ćemo to hopx). Pseudokodom bi njegov glavni, tj. najbitniji dio izgledao:

```
{

    Izračunaj d = korijen ( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) );

    Izračunaj hopx = (broj skokova između prve i druge točke) / d;

    Pošalji hopx na sva druga obojiva računala;

}
```

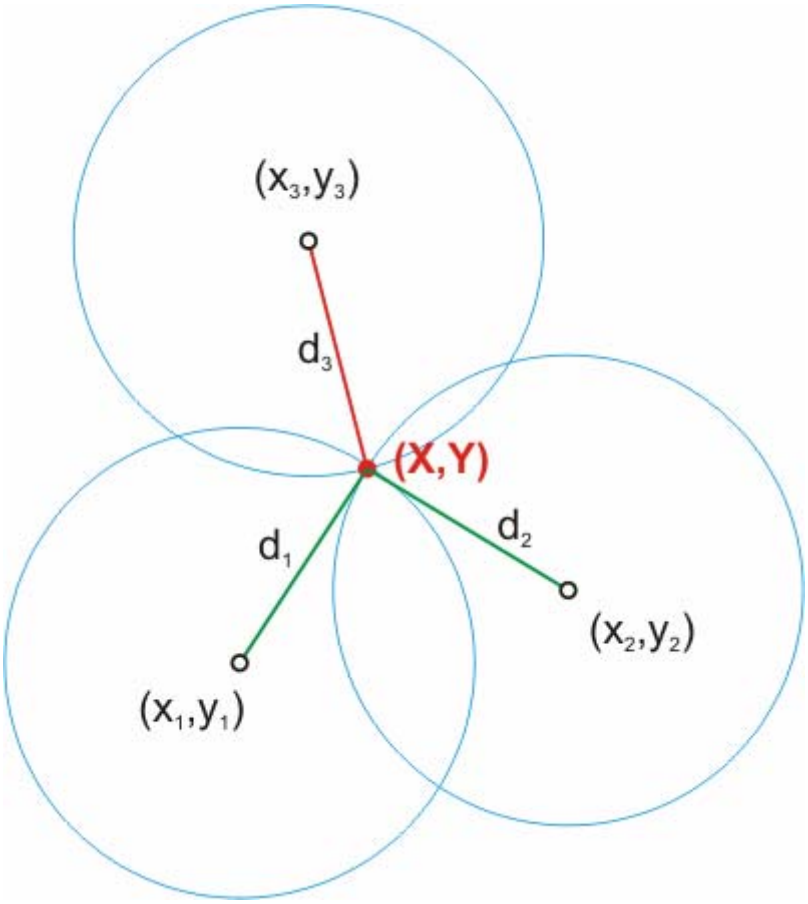
Osim toga, valja spomenuti način na koji treći dio ovog programa računa svoje koordinate. Doslovce rečeno, on pogađa. Ako imamo dvije kružnice koje se sijeku u dvije točke (pretpostavit ćemo da je korisnik bio dobre volje, ili ako nije, onda da je barem želio nacrtati koordinatni sustav na zidu), postoje 2 u potpunosti jednako dobra rješenja. Koje ćemo odabrati? Nama neće biti svejedno jer koordinatni sustav bojamo tako da mu je šaren jedino prvi kvadrant. Pa ako imamo dvije točke za koje vrijedi da je jedna unutar prvog kvadranta a druga ne, odabrat ćemo onu u prvom kvadrantu. Ako su obje izvan prvog kvadranta, izabrat ćemo onu bližu ishodištu. Konačno, ako su obje u prvom kvadrantu, odabrat ćemo onu dalju od ishodišta. Osim navedenoga, ovaj dio programa, kao i prethodna dva navedena dijela, ne radi ništa posebnije od širenja gradijentnog polja.




*Odabir „bolje“ točke*

## 7.3 Objašnjen rad programa Coordinate Calculator

Pri kraju opisa rada koordinatnog sustava, prvo ćemo nešto reći o bojanju. Ono se izvodi tako da se za cjelobrojne pozitivne koordinate točke (obojivog računala u kojem se nalazi taj program) izračuna zbroj tih koordinata po modulu broja raspoloživih boja u paleti i da dobiveni broj određuje redni broj boje za prikazivanje. Još preostaje da odredimo o kojim koordinatama se radi. Prethodno spomenuti program je napravio 3 gradijentna polja i to od 3 točke čije koordinate znamo; sveukupno: znamo koordinate 3 točke i udaljenost od svake. Zatim ćemo iskoristiti jednostavan matematički postupak pod nazivom trilateracija. Taj postupak zahtijeva postojanje 3 točke i poznate udaljenosti za dvije dimenzije, no, to možemo obaviti načelno i sa dvije točke i poznate udaljenosti, ali onda bi morali imati jasan kriterij odabira. U ovom programu je implementirano nešto što spada u obje kategorije. Čudno ili ne, tako je. Na početku ćemo izračunati koordinate dvije točke – presjecišta kružnica. Odabrat ćemo onu čija je udaljenost do treće točke „približnija“ udaljenosti do treće točke koju smo odredili preko gradijenata. Ne smijemo nikako računati na to da će nam se uvijek poklopiti vrijednosti i da neće nastati greška u računanju. Postupak je ilustriran na slici.




*Skica postupka trilateracije*

O programu Shadowfax kao i o njegovom malo poboljšanom bratu nazvanom Adaptive Shadowfax messenger nećemo ovdje govoriti. Razlog je taj što taj program sam po sebi se baš i ne uklapa u minijaturizam arhitekture obojivih računala. Pogled na izvorni kod tih programa bit će sasvim dovoljan za demonstraciju kompleksnosti. Ono što razdvaja ta dva programa je to što Shadowfax nije otporan na greške u komunikaciji, ali zato uvijek nudi pronalaženje relativno dobre staze, a Adaptive Shadowfax messenger je taj koji će na bilokoji način prenijeti poruku od jedne do druge strane, samo da ona stigne. Uostalom, izbjegavajte ovakve ogromne programe. Usporavaju simulaciju.

# 8 Literatura


- William Joseph Butera: Programming a Paintable Computer
- J.R.R. Tolkien: The Lord of the Rings
- Microsoft Developer Network: http://msdn.microsoft.com
- Wikipedia (Trilateration): http://en.wikipedia.org/wiki/Trilateration