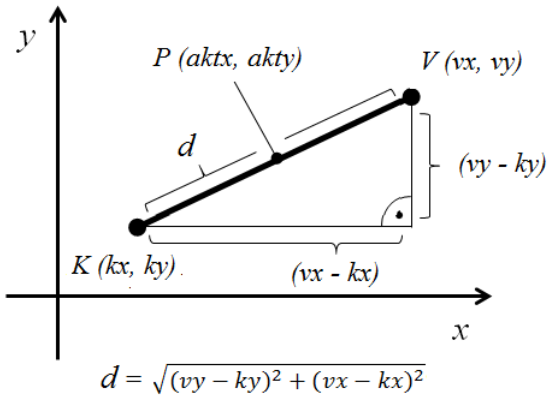
Mozgásvariációk

Két pont közötti egyenes vonalú egyenletes mozgás létrehozásához az kell, hogy a kezdő- és végpont között valamilyen lépésközzel végighaladjunk, és minden egyes köztes pont megrajzolása után ugyanannyit pihenjünk, és persze, frissítsük a rajzot.



A kezdőpont (K) és a végpont (V) között egy tetszőleges pont a P.

A két pont távolsága (d) Pitagorasz tétele alapján számolható.

Feladatunk a P pont koordinátáinak meghatározása és kirajzolása.

Ehhez mutatunk most három ekvivalens megoldást. Tanulmányozza át és értse meg őket, majd használja azt, amelyik a legjobban tetszik.

A példákban egy r sugarú pötty mozog a két pont között, de az alapelv nyilván más alakzatok mozgatásakor is ugyanez.

Az aktuális pötty kirajzolása:

```
public void rajzol(Graphics g){
    g.fillOval(aktx-r, akty-r, 2*r, 2*r);
}
```

A három mozgásrészlet:

1. Valamekkora lépésközzel végigmegyünk a két pont közötti távolságon, és a lépésköz által meghatározott lépésszám-szor kirajzoltatjuk a pöttyöt. Az arrébbmozdulás meghatározásához a vízszintes és függőleges távolságot is ennyi részre osztjuk. A frissítés a „szokásos”.

```
@Override
public void run() {
    double tavolsag = Math.sqrt((vy - ky)*(vy - ky) + (vx - kx)*(vx - kx));
    if (tavolsag > 0 && lepes > 0) {
        double lepesSzam = tavolsag/lepes;
        double dx = (vx-kx)/lepesSzam;
        double dy = (vy-ky)/lepesSzam;

        for(int i=0; i<= lepesSzam; i++) {
            try {
                aktx = (int) (kx + i * dx);
                akty = (int) (ky + i * dy);

                Thread.sleep(ido);
                frissites();
            } catch (InterruptedException ex) {
                Logger.getLogger(ZoldSzal.class.getName()).log(Level.SEVERE,
                                                                null, ex);
            }
        }
    }
}
```

2. Ez lényegében hasonló ötlet, csak nem `for`, hanem `while` ciklussal megyünk végig a távolságon:

```
@Override
public void run() {
    double tavolsag = Math.sqrt((vy - ky)*(vy - ky) + (vx - kx)*(vx - kx));
    if (tavolsag > 0 && lepes >0) {
        double aktualisTav = 0;
        while (aktualisTav < tavolsag) {
            try {
                aktx = (int) (kx + (vx - kx) / tavolsag * aktualisTav);
                akty = (int) (ky + (vy - ky) / tavolsag * aktualisTav);
                aktualisTav += lepes;

                Thread.sleep(ido);
                frissites();
            } catch (InterruptedException ex) {
                Logger.getLogger(KekSzal.class.getName()).log(Level.SEVERE,
                                                              null, ex);
            }
        }
    }
}
```

3. Ebben a változatban a szögfüggvények segítségével számolunk:

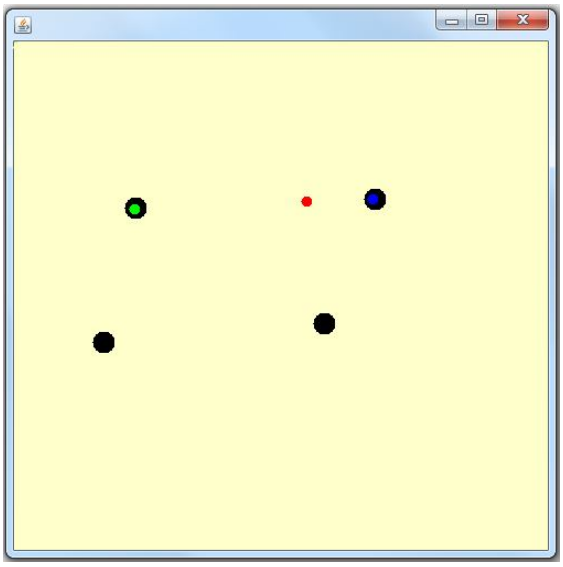
```
@Override
public void run() {
    double szog = Math.atan2(vy - ky, vx - kx);
    double tavolsag = Math.sqrt((vy - ky)*(vy - ky) + (vx - kx)*(vx - kx));
    if (tavolsag > 0 && lepes > 0) {

        for (double tav = 0; tav <= tavolsag; tav += lepes) {
            try {
                aktx = (int) (kx + tav * Math.cos(szog));
                akty = (int) (ky + tav * Math.sin(szog));

                Thread.sleep(ido);
                frissites();
            } catch (InterruptedException ex) {
                Logger.getLogger(PirosSzal.class.getName()).log(Level.SEVERE,
                                                                null, ex);
            }
        }
    }
}
```

Még egy megjegyzés a mozgásokkal kapcsolatban.

Ha már megkülönböztettük a háromféle szálat (1-zöld, 2-kék, 3-piros), akkor próbáljuk meg egymás után elmozgatni őket a megadott álló (fekete) pöttyök között. (Mondjuk, a fekete pöttyök a felületre való kattintáskor keletkeznek, és ha legalább kettő van belőlük, akkor közöttük egymás után elindulnak a kicsi színes pöttyök:



Valahogy így (most külön vezérlő osztály nélkül, csak a panelen dolgozunk):

```
    private void formMouseClicked(java.awt.event.MouseEvent evt) {
        int pottyR = 10;
        Potty potty = new Potty(evt.getX(), evt.getY(), pottyR);
        pottyok.add(potty);
        if(pottyok.size()>=2){
            int kx = pottyok.get(pottyok.size()-2).getKx();
            int ky = pottyok.get(pottyok.size()-2).getKy();
            int vx = pottyok.get(pottyok.size()-1).getKx();
            int vy = pottyok.get(pottyok.size()-1).getKy();
            int r = 5;
            double lepes = 2;
            long ido = 10;
            PirosSzal pirosSzal = new PirosSzal(kx, ky, vx, vy, r, lepes, ido, this);
            KekSzal kekSzal = new KekSzal(kx, ky, vx, vy, r, lepes, ido, this);
            ZoldSzal zoldSzal = new ZoldSzal(kx, ky, vx, vy, r, lepes, ido, this);
            szalak.add(pirosSzal);
            szalak.add(kekSzal);
            szalak.add(zoldSzal);
            pirosSzal.start();
            kekSzal.start();
            zoldSzal.start();
        }
    }
```

Csakhogy kellemetlen csalódás ér minket. A szálak ugyanis egyszerre indulnak, vagyis a színes pöttyök egymást takarva egyszerre mozognak, és nem egymás után.

Persze, rakhatnánk az indulások közé időzítőt, de van ennél jóval szebb megoldás, mégpedig az, hogy az úgynevezett Executor-t használjuk a szálkezeléshez. Ekkor némileg beleszólhatunk a szálütemező dolgába, és kicsit átvehetjük az irányítást. Pl. itt nézhet utána:

https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html

http://winterbe.com/posts/2015/04/07/java8-concurrency-tutorial-thread-executor-examples/

http://tutorials.jenkov.com/java-util-concurrent/executorservice.html

Az ilyen elvű módosítás:

```
private void formMouseClicked(java.awt.event.MouseEvent evt) {
    int pottyR = 10;
    Potty potty = new Potty(evt.getX(), evt.getY(), pottyR);
    pottyok.add(potty);
    if(pottyok.size()>=2){
        int kx = pottyok.get(pottyok.size()-2).getKx();
        int ky = pottyok.get(pottyok.size()-2).getKy();
        int vx = pottyok.get(pottyok.size()-1).getKx();
        int vy = pottyok.get(pottyok.size()-1).getKy();
        int r = 5;
        double lepes = 2;
        long ido = 10;

        PirosSzal pirosSzal = new PirosSzal(kx, ky, vx, vy, r, lepes, ido, this);
        KekSzal kekSzal = new KekSzal(kx, ky, vx, vy, r, lepes, ido, this);
        ZoldSzal zoldSzal = new ZoldSzal(kx, ky, vx, vy, r, lepes, ido, this);
        szalak.add(pirosSzal);
        szalak.add(kekSzal);
        szalak.add(zoldSzal);

        //      pirosSzal.start();
        //      kekSzal.start();
        //      zoldSzal.start();

        ExecutorService executorService=Executors.newSingleThreadExecutor();
        executorService.execute(kekSzal);
        executorService.submit(pirosSzal);
        executorService.execute(zoldSzal);
    }
}
```