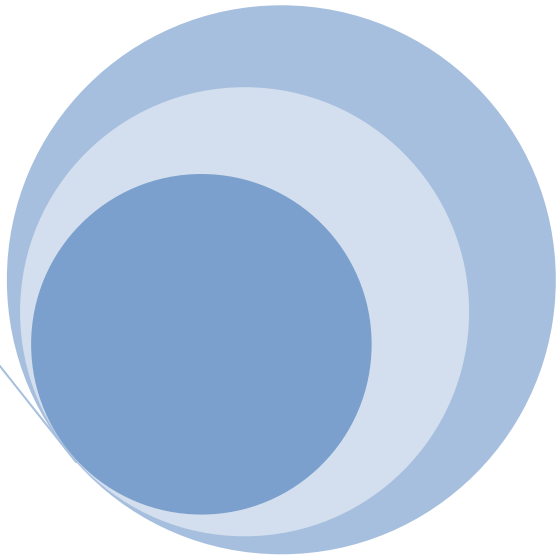
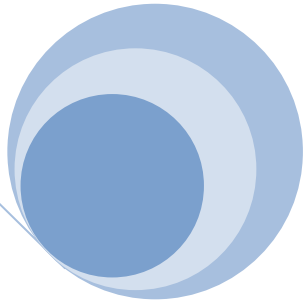
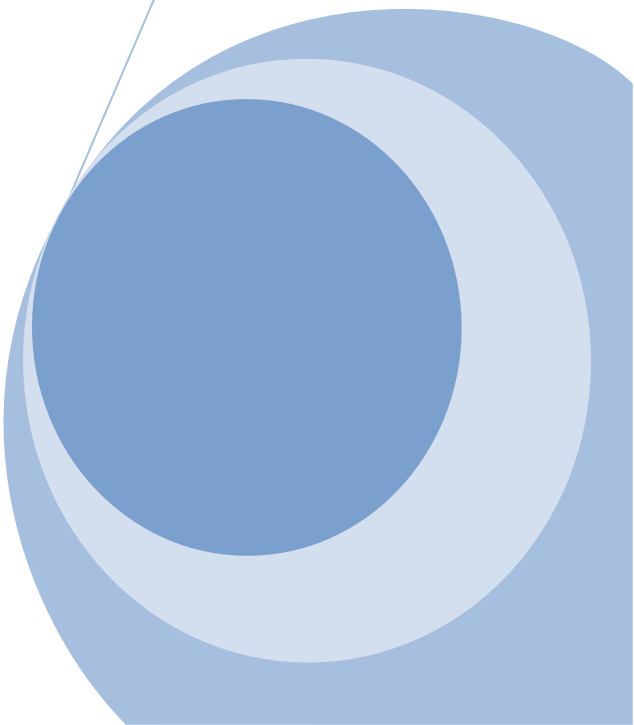






AI Flocking

Artificial Intelligence

Simon Karman - 500621839
Simon Karman - 500621839

Simon
19-3-2012

Vraag 1:

Zorg dat de Units elkaar volgens de drie regels van Flocking beïnvloeden.

Alignment:

Zodra een Unit meer dan 0 buren heeft moet die een beetje van die buren af bewegen. Dit kan je bereiken door eerst de hoek tussen elke buur en zichzelf de berekenen. Vervolgens hier een gemiddelde van te nemen en een beetje in die richting te bewegen.

```
@Override
public Vector2 calculate( Unit self, List<Unit> neighbors )
{
        Vector2 result = Vector2.ZERO;

        //Zijn er meer dan 0 buren
        if (neighbors.size() > 0)
        {
            //Loop door alle buren heen
            result = self.getDirection();
            for (int i =0; i < neighbors.size(); i++) {
                //Voeg de direction van elke buur toe aan deze direction
                //Self zal dus van elke buur een beetje de zelfde richting krijgen.
                result = result.add(neighbors.get(i).getDirection());
            }
        }

        return result;
}
```

Cohesion

Zodra een Unit meer dan 0 buren heeft moet die een beetje van die buren af bewegen. Dit kan je bereiken door eerst de hoek tussen elke buur en zichzelf de berekenen. Vervolgens hier een gemiddelde van te nemen en een beetje in die richting te bewegen.

```
@Override
public Vector2 calculate( Unit self, List<Unit> neighbors )
{
        Vector2 result = Vector2.ZERO;

        //Zijn er meer dan 0 buren
        if (neighbors.size() > 0)
        {
            //Loop door alle buren heen
            for (int i =0; i < neighbors.size(); i++) {
                //Bereken het verschil in x en y richting tussen de buur en zichzelf
                Vector2 xdif = new Vector2(   neighbors.get(i).getPosition().x - self.getPosition().x, 0);
                Vector2 ydif = new Vector2(0, neighbors.get(i).getPosition().y - self.getPosition().y);
                //Bereken de hoek tussen die 2 vectoren
                float angle = (float) Math.acos(ydif.normalized().dot(xdif.normalized()));
                Vector2 angleVector = new Vector2(Math.sin(angle), Math.cos(angle));
                //Voeg dit bij de result erbij. Elke buur zorgt dus ervoor dat self iets meer naar de buur toebeweegt.
                result = result.add(angleVector);
            }
        }

        return result;
}
```

Seperation

Zodra een Unit meer dan 0 buren heeft moet die een beetje van die buren af bewegen. Dit kan je bereiken door eerst de hoek tussen elke buur en zichzelf de berekenen. Vervolgens hier een gemiddelde van te nemen en een beetje in de overgestelde richting te bewegen. Dit om een beetje van de buren weg te komen.

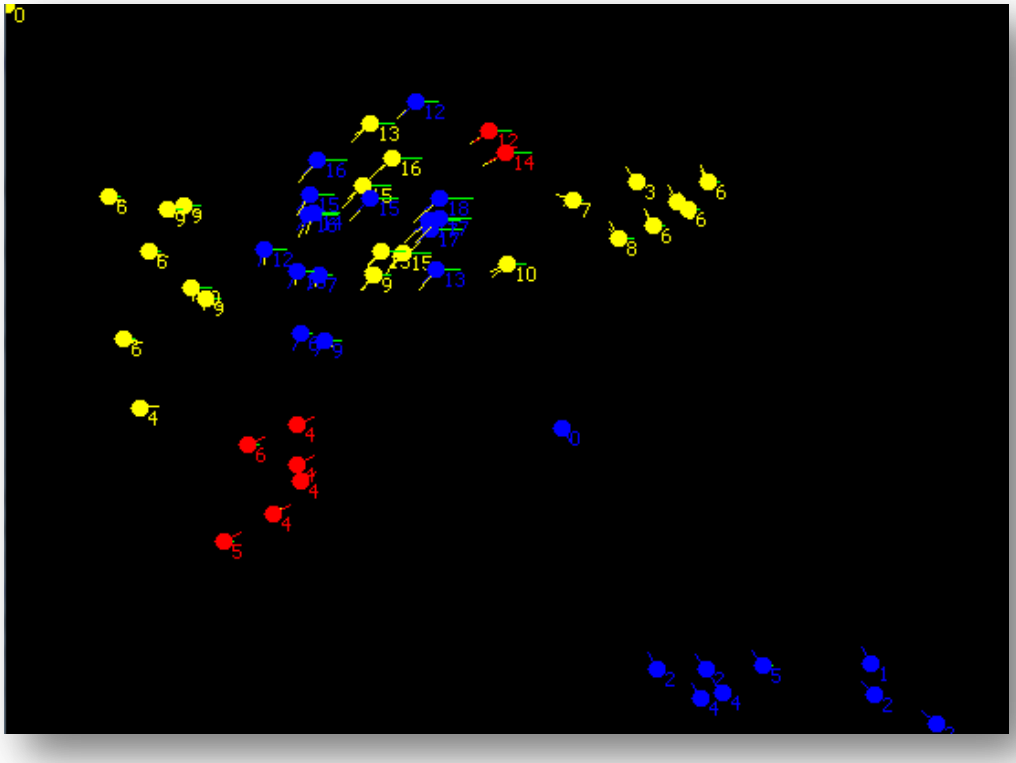
```
@Override
public Vector2 calculate( Unit self, List<Unit> neighbors )
{
        Vector2 result = Vector2.ZERO;

        //Zijn er meer dan 0 buren
        if (neighbors.size() > 0)
        {
            //Loop door alle buren heen
            for (int i =0; i < neighbors.size(); i++) {
                //Bereken het verschil in x en y richting tussen de buur en zichzelf
                Vector2 xdif = new Vector2(   neighbors.get(i).getPosition().x - self.getPosition().x, 0);
                Vector2 ydif = new Vector2(0, neighbors.get(i).getPosition().y - self.getPosition().y);
                //Bereken de hoek tussen die 2 vectoren
                float angle = (float) Math.acos(ydif.normalized().dot(xdif.normalized()));
                Vector2 angleVector = new Vector2(Math.sin(angle), Math.cos(angle));
                //Voeg dit bij de result erbij. Elke buur zorgt dus ervoor dat die iets meer van die buur afbeweegt.
                result = result.add(angleVector).scale(-1);
            }
        }

        return result;
}
```

Voorbeeld

Hier een plaatje waarin goed te zien is hoe de verschillende Units elkaar beïnvloeden.



Vraag 2

Bij vraag 2 was het te bedoeling om te zorgen dat Units met dezelfde kleur elkaar meer te beïnvloeden.

Dit is dus goed te bereiken door te zeggen dat Units met de zelfde kleur elkaar meer aantrekken en Units met een verschillende kleur elkaar beter afstoten. Ook moeten Units met de zelfde kleur meer van elkaar de zelfde richting krijgen.

Met de volgende regels code is dit in elke rule (Alignment, Cohesion en Seperation) te bereiken:

```
if (self.type.color == neighbors.get(i).type.color) {
    result = result.add(angleVector.scale(-1));
}
result = result.add(angleVector.scale(-0.1));
```

Test

Hieronder een test waarin goed is te zien dat de zelfde kleuren het meest bij elkaar blijven.