Peer-Review 4
Objektorienterad programmering med Java

Emil Björklund - embj3739
emilbjorklund@live.com

Källkod: Tobias Isaksson

May 7, 2019

Inledning

- IDE: Visual Studio Code
- Java version: JDK 8 update 191
- OS: MacOS Mojave

Peer Review

Programvarutestning

Programmet kunde exekveras utan några felmeddelanden. Med hjälp av rapportens guide kan användaren lätt förstå strukturen. Användaren får själv kompilera programmet och därefter exekvera **Param** för manipulering av värden eller **Pasture** för att använda sig av defaultvärden.

Kravvalidering

- **Parametrarna skall kunna anges när simuleringen påbörjas.**
  Parametrar kan ändras vid start, dock mindre praktiskt att användaren behöver starta två olika program beroende på om parametrar skall manipuleras eller ej.
- **Alla parametrar ska ha defaultvärden.**
  Alla parametrar har defaultvärden dock utrotas ofta någon art relativt fort. Det problematiska hurvida en entitets startpossition slumpas fram kan det vara svårt att skapa balans.
- **Det ska vara möjligt att ändra enstaka parametrar.**
  När användaren ska ändra parametrar anropas programmet **Param** i dialogrutan för parametrar är defaultvärderna redan ifyllda vilket resulterar i att användaren kan ändra endast önskade parametrar. Dock är det inte möjligt att radera ett fält och lämna det tomt.
- **Med defaultparametrar ska simuleringen vara någotsånär stabil.**
  Det blir olika resultat av simuleringen vilket i viss mån är en typ av balans.
- **Invånarna har ingen magisk vetskap om hagens storlek.**
  Invånarna använder sig av en metoden isCompatible() för att verifiera om en viss punkt är tillänglig.
- **Simuleringsregler.**
  I klassen Plant finns det en metod isCompatible() som undersöker hurvida objektet av typen Plant får skapas på en viss punkt. Enligt den metoden får objekt av denna typ skapas där det existerar objekt av typen Sheep eller Wolf. Detta resulterar i att vissa föremål tillfälligt målas över tills nästa iteration. I övrigt följer simuleringen givna regler.
- **Programmet bör vara fritt från duplicerad kod.**
  Duplicering av kod förekommer på vissa ställen i källkoden.

Kodstruktur

Koden är väl dokumenterad vilket gör det enkelt att analysera den. På vissa ställen i kommentarsblocken är användingen av @param inte helt korrekt. @param skall direkt följas av variabelnamnet som finns i metodhuvudet för att sedan ges en beskrivning.

Den objektorienterade strukturen har ett bra upplägg med en logisk klasshierarki. Angående duplicering av kod finns det några punkter där det finns utrymme för optimering.

```
private final ImageIcon image = new ImageIcon("sheep.gif");     // Denna Entitys bild
private final Pasture pasture;    // Var finns denna Entity
private final int duplicateRate;  // Med vilket intervall skall entity föröka sig
private final int ageDuplicate;   // Hur gammalt skall entity vara för att föröka sig
private final int entitySpeed;      // Hur många tick mellan förflyttningar
private final int maxTimeNoEat;      // Hur länge klarar sig entity utan mat
private final int lengthOfVision; // Hur långt ser entity
private int age;    // entityns ålder
private int speed;   // Hur fort rör sig entity (lägre siffra = snabbare).
Antalet tick mellan förflyttningar.
private int lastEat;  // När åt entity senast?
private boolean hasEaten = false;  // Har denna entity ätit någon gång?
```

Kodstycket ovan är hämtat från klassen Sheep, klassen Wolf har snarlika klassvariabler. Klassen Moving skulle kunna utöka sin funktionalitet genom att ändra dess funktion. Om moving (med ett annat namn) representerar levande djur skulle koden ovan vara generisk för alla djur i simuleringen. Alla djur rör sig, äter, har en mognadsålder, en levnadsålder osv. därför skulle det vara praktiskt att samla det i en superklass för att sedan ha separata klasser för olika typer av arter som definierar dess specifika unika karakteristik. Variablerna image som innehåller information angående bilden som ritas upp samt variabeln som refererar till Pasture kan flyttas till klassen Entity då det är gemensamt för alla objekt i simuleringen. Klassen Moving är av typen abstrakt dock definierar inte klassen några abstrakta variabler eller metoder. I den abstrakta klassen Entity definieras en abstrakt metod tick() denna implementeras inte i klassen Moving som ärver från Entity. Dock implementeras metoden i underklasserna Sheep och Wolf därmed skulle metoden Moving inte behöva vara abstrakt om det implementeras en alternativ lösning för metoden tick(). Namngivningen av metoder och variabler följer samma mönster och är beskrivande för ändamålet.

Slutord

Rapporten är mycket väl skriven vilket gör det enkelt att förstå motivationen bakom programmets implementation. Gällande att benämna djurens beteende som artificiell intelligens håller jag inte med. A.I uppnås ofta genom machine learning samt neurala nätverk där en dator med hjälp av träningsdata kan lära sig vilka beslut som leder till bäst resultat. Ett tränat nätverk med optimerade viktade noder kan sedan utan vidare input fatta beslut inom det område som nätverket är tränat på. Att basera rörelse på slumpmässiga integers är en bra strategi vilket även jag har implementerat som lösning men jag skulle inte benämna det som A.I. I det stora hela är det en mycket bra lösning.