Eksamensopgave

Objektorienteret Programmering og Design (OOPD)
2013 - 2014, blok 2

David Grynnerup Pedersen (tlb209)

Designet

Jeg valgte at skrukture min løsning efter MVC mønstret. Dvs. jeg har modeller som holder står på kerne objekternes interne state og sikre at alle reglerne for simulationen bliver overholdt. Så har jeg views som viser modellerne. I mit tilfælde har jeg kun ascii terminal views. Til sidst jeg har en controller som starter simulationen, ber modellerne om at opdatere sig selv og renderer views.

Jeg valgte at dele model laget op i følgende hovedklasser med følgende ansvar:

- Environment:
  - Holder styr på koordinatsystemet som dyrene bevæger sig rundt i.
  - Flytter dyrene til deres nye positioner.
  - Fjerner døde eller spiste mus.
- Point:
  - Et punkt i koordinatsystemet.
  - Indkapsler også diverse hjælpe funktioner som modellerne bruger.
- Square:
  - Et felt i koordinatsystemet.
  - Holder styr på en liste af ting og sørger for ting ikke bliver placeret steder hvor det ikke er tilladt.
  - Bestemmer om mus kan reproducere sig selv her.
  - Bestemmer om den indeholder mus som kan spises.
  - Har også metoder som viewsne bruger til at vise miljøet.
- Mouse:
  - Holder styr på musens liv.
  - Beslutter hvor den gerne vil gå hen.
  - Beslutter om den vil reproducere sig selv.
- Owl:
  - Beslutter hvor den gerne vil gå hen.
- Stone:
  - Blot en dum sten.

Yderligere har et par små service klasser som fungerer som hjælpe klasser til modellerne.

Selve koordinatsystemet valgte at lave som et HashMap af punkter og felter. Dvs miljøet holder styr på felterne og deres koordinater. Feltet ved ikke selv hvor det er i verden, og eftersom feltet holder styr på dyrene, ved de heller ikke hvor de er i verden. Jeg valgte at strukturerer mit system på en sådan måde da det giver mig bedre mulighed for at fordele logikken over flere klasser i stedet for at have få klasser med en masse ansvarsområder.
Helt konkret betyder det at for at mus og ugler kan bevæge sig rundt i miljøet bliver de givet deres position lige nu same det miljø de befinder sig i. Så laver de selv en masse beregninger og returnerer det punkt de kan og vil gå hen til. Det er altså miljøet der flytter tingene rundt, de flytter ikke sig selv.

Diverse overvejelser og beslutninger

Jeg valgte at bruge engelske navne til alt undtagen de klasser der er nævnt i opgavebeskrivelsen.

Når et felt indeholder en spiselig mus er det den mus der ankom først på feltet som bliver spist. Dvs at indeholder et felt to mus og en sten er det sådan den mus som ligger under stenen der bliver spist.

Ugler og mus kan ikke starte på samme felter. Kunne det ske ville musen blive spist med det samme og simulationen starter dermed ikke med 150 mus.

Jeg har ikke taget højde for hurtig min kode kører på nogen måde. I stedet har jeg fokuseret på at bryde systemet op i flere små overskuelige dele.

Jeg valgte at ugler spiser musen på det felt de lander inden de egentlig lander der. Dvs. at det er lovligt for en ugle at beslutte sig for at lande på et felt der indeholder to mus, da den ene mus vil blive spist med det samme.

Logikken for hvornår det er tilladt for en ting at starte på et felt og hvor det er tilladt for en ting at bevæger sig en til et felt er dermed forskellig. Dette lavede jeg service klasser der indkapsler. Disse klasser bliver så brugt at Miljø og Square.

Kørsel af simulation

Efter 10 skridt:

```
00031003300300300070
73300003003373370000
00303000000303000730
00730030300377000300
01030730010033000000
00007006600030303070
03330300330300003330
00000000007013000033
70330030307000030303
00030333007703073003
03335030003300000030
00000000000003330300
30300030303700003030
07030000070000000373
05730000007000003000
00700037073030000003
33010030005303023000
03010300000000000033
00003077077300030000
00070300000003100000
Current number of mice: 164
Number of mice eaten: 9
```

Efter 50 skridt:

```
00001000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
01000000010000000000
00000000000000000000
00200000000000000000
00000000000040000000
00000000000000000000
00000000000000000000
00001000000000000000
00000000000000000000
00000000000000000000
00000000000000000000
01000000000000000000
00000000000000000000
00010000001000000000
00010000000000000000
00000000000000000000
00000000000000100000
Current number of mice: 0
Number of mice eaten: 20
```

Kørsel af unit tests

```
JUnit version 4.11
..............................................................................
......
Time: 4.367

OK (86 tests)
```

Kommentarer til kørsler

Jeg primært lavet test dreven udvikling så har derfor en del tests som kan bruges til at sikre at hele systemet fungerer. Jeg har primært valgt at lave integrations tests da mulighederne for at lave isolerede tests i java er meget bøvlede. Havde mulighederne været bedre ville jeg have gjort brug af ca 90% unit tests og 10% integrations tests (ved brug af mocks og stubs). Det giver en test suite som for det første er meget hurtig (dette er vigtigt i TDD) men også giver en suite som giver meget præcis feedback om hvor systemet ikke fungerer. Sådan feedback får man ikke ved kun at lave integrations tests.

Vi ser at musene er forholdsvis gode til at undvige uglerne da der efter 50 skridt kun et spist 20 mus. Dog er der slet ikke nok mus til at de kan holde bestanden oppe og efter de første 150 mus er døde har de ikke nået at formere sig nok til at kunne komme tilbage i kampen.