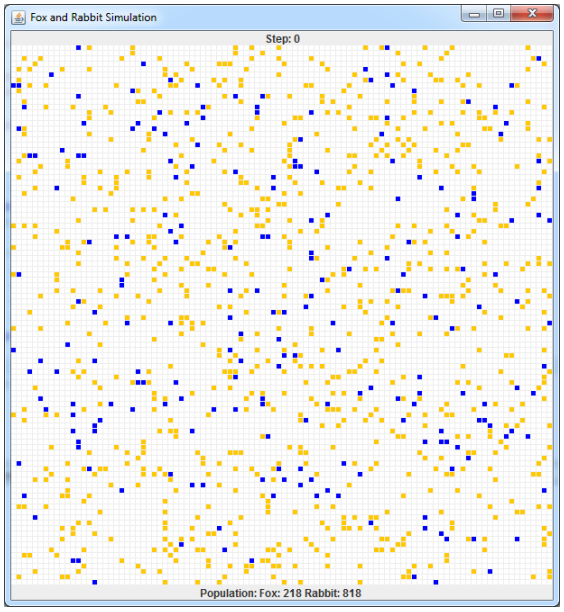
Aufgaben

Kapitel 10.1, 10.2.2 und 10.2.3

1. zu bearbeitende Aufgabe:
   - Öffnen Sie das Projekt foxes-and-rabbits-v1.
   - Studieren Sie den Source-Code der Klassen Rabbit, Fox und Simulator.
   - Erzeugen Sie einen Simulator ‚Silberweid'.
   - Führen Sie mehrere Simulationen unterschiedlicher Anzahl Schritte durch.

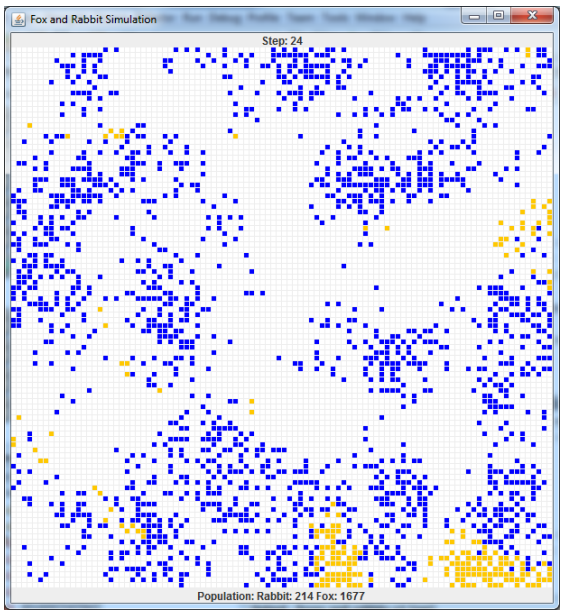
   Was fällt Ihnen dabei auf?

   Die Ausgangslage ist zufällig (resp. Random wird statisch initialisiert), ob dies wirklich der Realität entspricht ist zu bezweifeln.

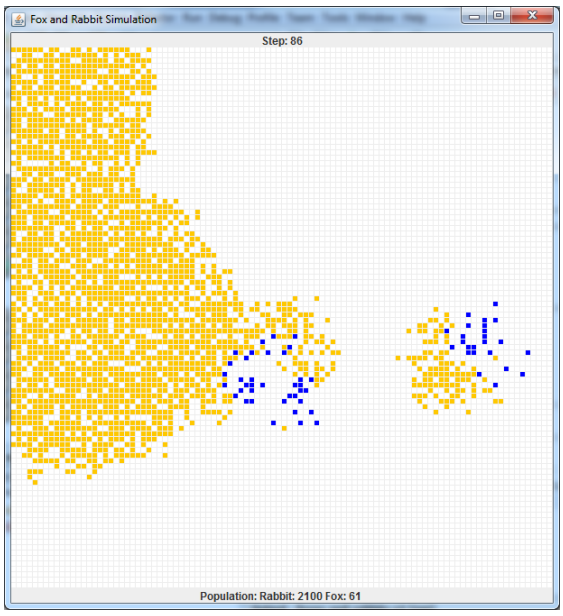


Die Population zwischen Hasen und Füchse ist ein Wechselspiel.

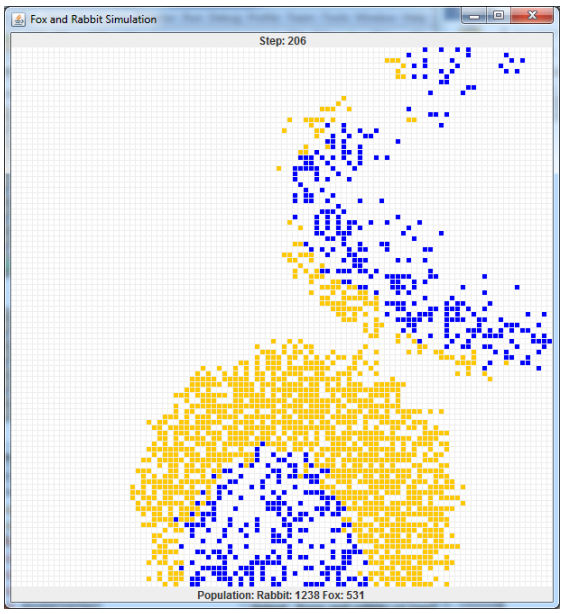
Überpopulation von Füchse:



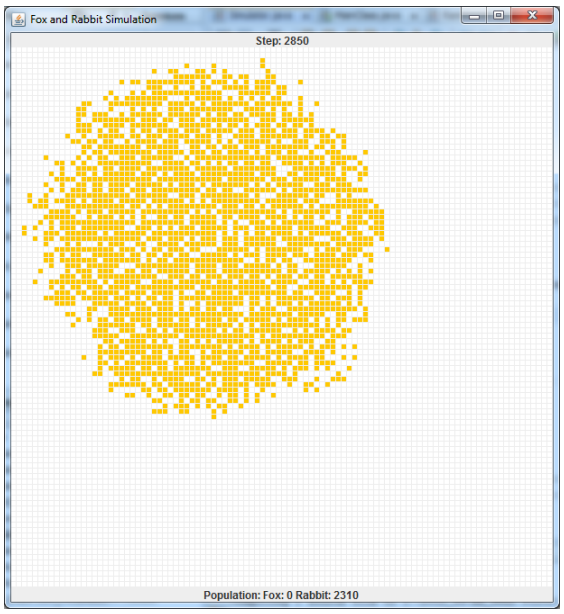
Überpopulation von Hasen:



Auch wenn die Hasen sich schneller vermehren, die Füchse fressen dafür umso mehr…



Je nach „Feldgrösse“ Überlebt der Hase und der Fuchs stirbt aus.



2. Was für Vorteile bringt uns das Konzept der Simulation?
   - *Es können „unmögliche“ und gefährliche Experimente durchgeführt (resp. eben simuliert) werden.*
   - *Günstiger, einfacher und schneller als in Real, z.B. Warteschlagen, Stau (Strassenverkehr)*
   - *Messwerte berechnen, z.B. Umweltveränderungen, globale Erwärmung, Wetterprognosen*

3. Hat die Simulation auch Nachteile?
   - *Es wird nur ein Teil betrachtet und nicht das ganze „System“.*
   - *Je nach Detailtreue kann es ungenau / verfälschte Resultate liefern*
   - *Ressourcen: Rechenleistung, Simulationsdauer, Speicher*
   - *Die verwendeten Modelle sind Vereinfachungen der Realität, dadurch ggf. ungültige Ergebnisse*

4. Identifizieren Sie im Code die in der Vorlesung behandelten Schwächen.
   done

5. Lösen Sie die Aufgaben 10.9 und 10.11 bis 10.15 im Buch.

10.9:

```
// Determine whether a shared random generator is to be provided.
    private static final boolean useShared = true;
```

Wenn ein unterschiedlicher Random generator verwendet wird, resp. dieser nicht mit einem definierten Seed initialisiert, kann die Simulation nicht wiederholt werden.

10.11:

Die Tiere haben kein Geschlecht, alle können Junge bekommen. Die Ausbreitungsgeschwindigkeit der Tiere ist ebenfalls statisch. In der Natur wäre diese wohl unterschiedlich, sowie würde sich das Verhalten ändern wenn z.B. der Hase gejagt wird.

10.12:

Wenn es mehr Hasen gibt, haben die Füchse mehr zu essen und vermehren sich dadurch schneller. Soweit bis es zu viele Füchse sind und sie somit die Hasen aufessen, ausrotten. Somit sterben dann auch die Füchse weil sie nichts mehr zu fressen haben. Wenn es zu wenig Nachwuchs bei den Hasen gibt, werden diese aufgefressen und die Füchse sterben ebenfalls infolge mangelnder Nahrung aus.

10.13:

Die „Suche“ eines Fuchses wäre in Realität wohl über ein grösseres Feld verteilt ans in der Implementation. Ebenfalls wird nicht beachtet wie alt ein Hase ist, falls er gefressen wird. Ein junger Hase würde wohl nicht gleich viel Energie liefern wie ein älterer.

10.14:

Beides ist richtig. Wenn die Füchse älter werden, wird es eine grössere Population von ihnen geben. Dadurch werden auch mehr Hasen gefressen. Somit wird indirekt der Hase ausgerottet.

10.15:

Je nach Parameter wird der Fuchs oder Hase viel schneller aussterben. Die „Default“ Parameter sind sehr gut ausgewogen und wohl die am besten gewählten.

6. Lösen Sie die Aufgaben 10.20 bis 10.29 und 10.31 bis 10.38 sowie 10.40 im Buch.

10.20:

Sobald die Füchse genug alt sind, wird sich das ganze wieder „einpendeln“. Ein Zufalls Alter ist nicht wirklich Wahrheitsgetreu – jedoch wird die Simulation dadurch nicht wirklich beeinflusst.

10.21:

Beim Methoden Aufruf wird jeweils eine List als Parameter übergeben, worin die neuen Füchse gespeichert werden. Zusätzlich wird mittels der isAlive() Methode abgefragt ob der Fuchs gestorben ist. Gleiches verhalten gilt auch für die Hasen.

10.22:

Die Listen müssten nicht synchronisiert werden, jedoch würde wohl ein neu erstellen der gesamten Listen vor jedem Simulationsschritt mehr Rechenleistung in Anspruch nehmen.

10.23:

Theoretisch: startwerte definieren, bestimmte Anzahl Simulations-Schritte ausführen und danach die jeweiligen Arrays auf ihren Inhalt überprüfen.

10.24:

Variablen:

- *age*
- *alive*
- *location*
- *field*

Konstanten: (gleicher Name, jedoch unterschiedliche Werte !!!)

- *BREEDING_AGE*
- *MAX_AGE*
- *BREEDING_PROBABILITY*
- *MAX_LITTER_SIZE*

Methoden:

- *run() / hunt () (unterschiedlicher Name, aber grundsätzlicher wird darin das Verhalten abgebildet)*
- *isAlive()*
- *setDead()*
- *getLocation()*
- *setLocation()*
- *incrementAge()*
- *giveBirth()*
- *breed()*
- *canBreed()*

Konstruktor:

- *Die Konstruktoren sind mehrheitlich gleich, Fox ist etwas erweitert.*

10.25:

- *isAlive()*
- *setDead()*
- *getLocation()*
- *setLocation()*
- *canBreed()*
- *giveBirth()*
- *breed()*
- *incrementAge()*

10.26:

Nein, sie haben nur unterschiedliche Werte.

10.27:

Blackbox-Testing

10.28:

done

10.29:

Code-Duplication wurden entfernt. Es kann nun relativ einfach ein weiteres Tier hinzugefügt werden. Code ist „einfacher" geworden, übersichtlicher.

10.31:

Ja, sobald es mindestens eine abstrakte Methode gibt, muss die Klasse als abstrakt definiert werden.

10.32:

Ja, dies ist möglich. Es kann dann einfach kein Objekt von dieser Klasse erstellt werden.

10.33:

Ja, dies kann Sinn machen. Z.B. wenn absichtlich kein Objekt davon erstellt werden darf und die entsprechenden Methoden zwingend implementiert werden müsse von den jeweiligen Subklassen.

10.34:

Durch das Keyword „Interface".

10.35:

Ja – die abstrakten Methoden müssen implementiert werden von der Subklasse.

10.36:

Damit entsprechende Anpassungen in den Subklassen vorgenommen werden können, ist es wichtig zu wissen wie @Override funktioniert.

10.37:

Grundsätzlich möglich – dies wäre ja dann einfach eine eigene Klasse mit nur einer Methode.
Ob dies wirklich Sinn macht? Eine eigene Methode in der bestehenden Klasse würde ja +/- zwecks Code „Vereinfachung" den gleichen Effekt haben.

10.38:

done

10.40:

done

Handout DAT2 (Datenstrukturen)

7. Beantworten Sie die Kontrollfragen A, B und C.
   Halten Sie allfällige Unklarheiten und Fragen schriftlich fest.
   Siehe PRG2_DAT2_KF.docx / PRG2_DAT2_KF.pdf