Thomas Favre-Bulle
André Ourednik
Semaine ENAC
28 Avril 2014

Le jeu de la vie de Conway

Introduction à la plateforme Netlogo

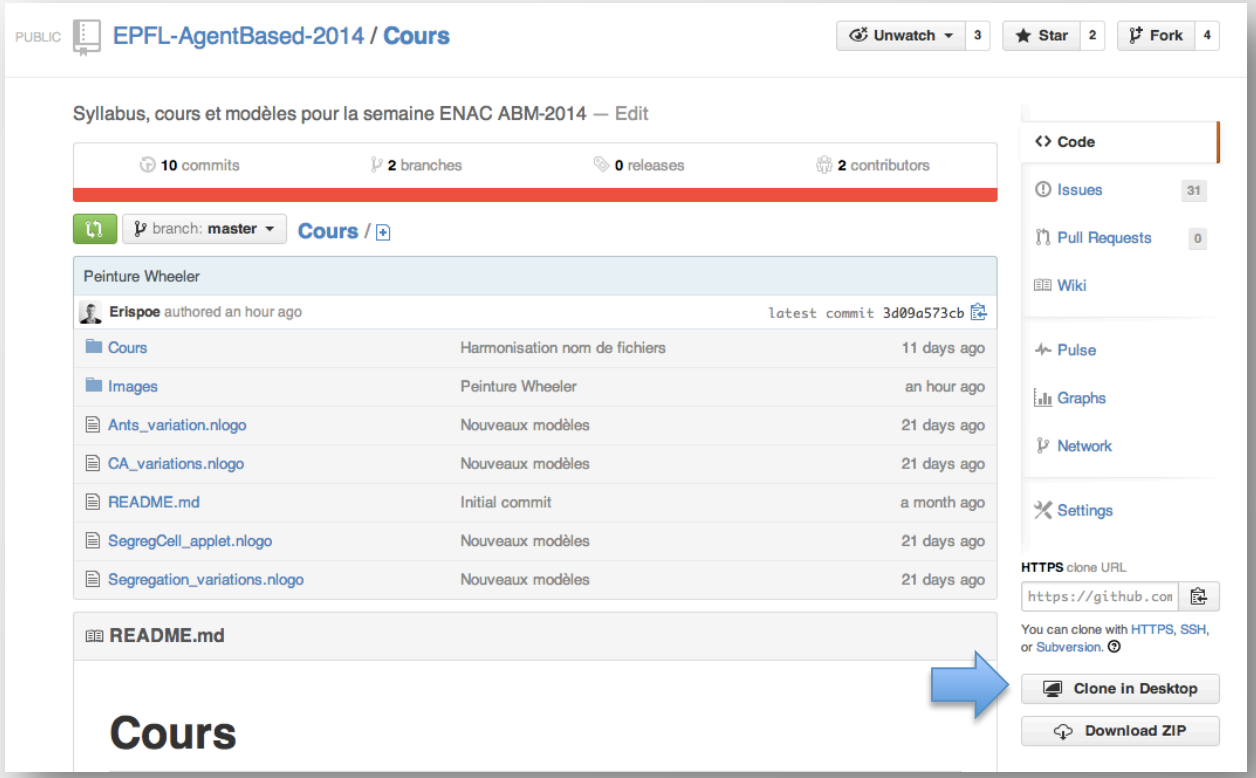
Partie 1

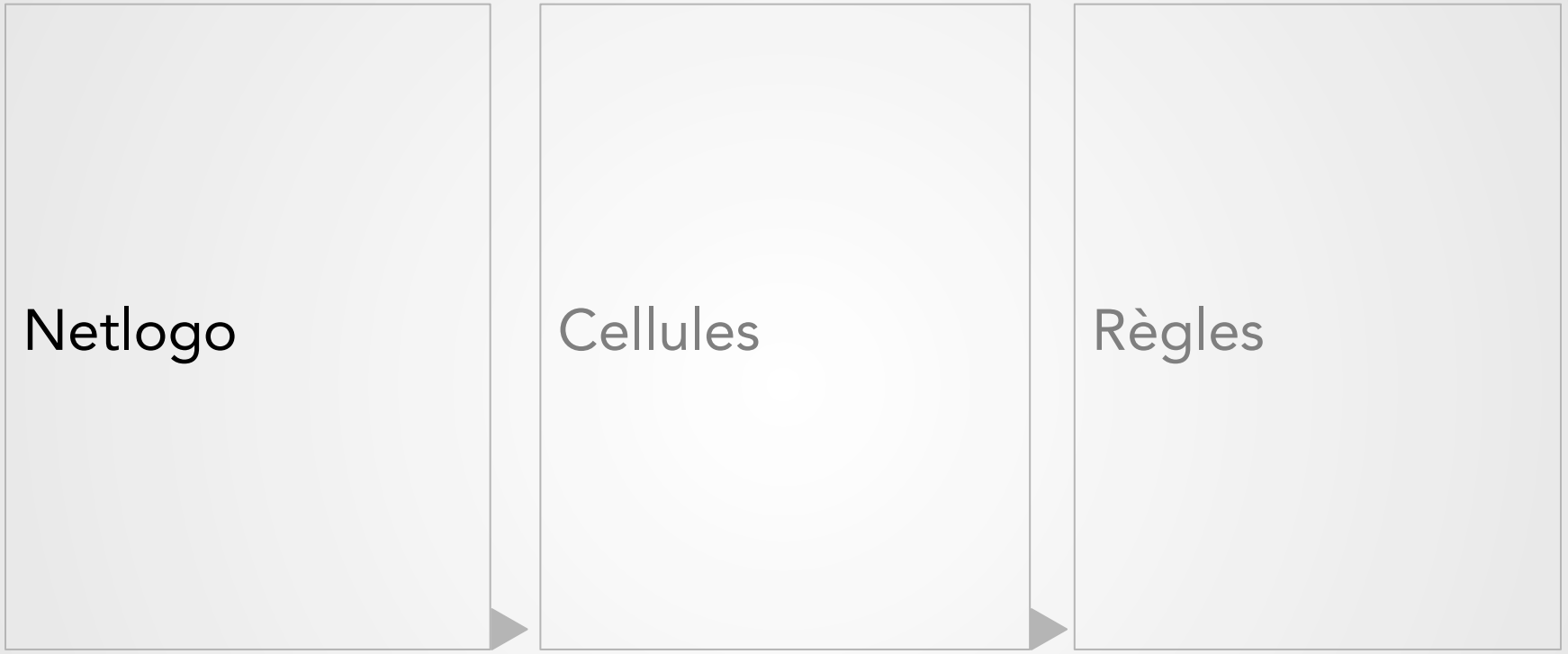
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Faculté de l'Environnement Naturel, Architectural et Construit
École Polytechnique Fédérale de Lausanne

Cloner un dépôt GitHub

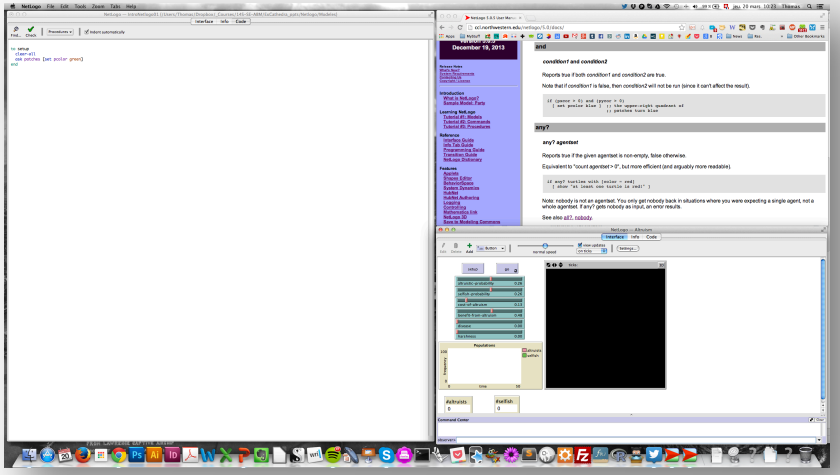
- Cloner un dépôt permet d'en avoir une copie locale.
- Un dépôt cloné est mis à jour.



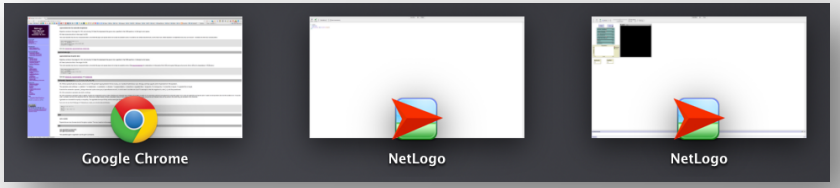

Netlogo
Cellules
Règles

Trouvez votre configuration de travail

- Vous aurez besoin de:
  - Votre code (instance Netlogo)
  - La documentation Netlogo (web), toujours ouverte
  - Un code de référence (seconde instance Netlogo)
- Réflexe primordial: toujours regarder dans la documentation en cas de doute.
- Votre configuration de travail influe sur votre productivité et sur votre confort.
- Faites des tests et changez:
  - Une fonction par bureau
  - Grand écran mosaïque
  - ...



Grand écran



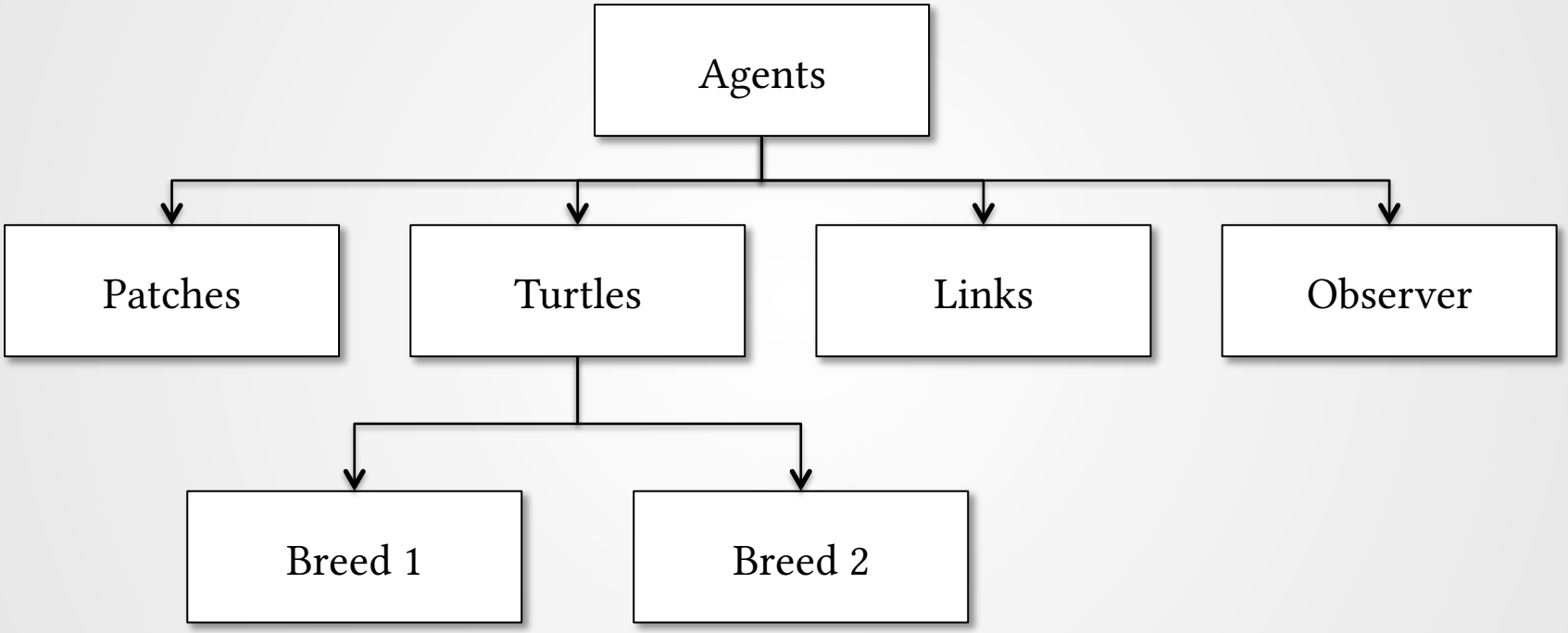
Plusieurs bureaux

Commentaires

- Les commentaires sont essentiels pour:
  - Vous: retravailler le code après une période d'inactivité
  - Les autres: comprendre ce que le code fait et comment il le fait
- Ne vous reposez jamais sur votre compréhension implicite du code.
- Mieux vaut commenter trop que pas assez.

```
;; Ceci est un commentaire
;; Chaque ligne commence par
;; des points-virgules
```

Agents



Les agents sont des classes

- Les classes décrivent des propriétés génériques des objets.
  - Paramètres
  - Procédures
  - Ex: chaque *turtle* a une coordonnée x et y
- Chaque instance d'une classe (chaque objet) peut avoir des valeurs différentes
  - Ex: chaque *turtle* peut avoir des coordonnées x et y différentes
  - Ex: chaque *patch* peut avoir un état différent (mort ou vivant, 0 ou 1, couleur…)

Netlogo

Cellules

Règles

De quels éléments aurons-nous besoin?

- Agents créés par Netlogo:
  - Observer
  - Patches = cellules
    - Vivant / white / 1
    - Mort / black / 0
- Règles de transition
  - Nombre de voisines vivantes pour naître
  - Nombre de voisines vivantes pour survivre
  - Nombre de voisines vivantes pour mourir

Variables globales

- Variables d'environnement
  - paramètres globaux du modèle
    - ex: pourcentage de cellules vivantes à l'initialisation
  - grandeurs statistiques mises à jour par des procédures
    - ex: pourcentage de cellules vivantes
- Déclarées:
  - au début du code
  - dans l'interface graphique (déclaration implicite)
- Accessibles à tous les agents.

```
globals [
    variable1
    variable2
]
```

Types de variable

- Accessibilité
  - globale: accessible partout
  - locale: accessible uniquement dans le bloc de code dans laquelle elle est définie
- Type
  - numérique
  - string (chaîne de caractères)
  - booléenne (true, false)
  - extensions: array, matrix…

```
globals [
    variable1
    variable2
]

to procedure
    let x 3
    set x 4
end
```

Variables de classe

- Tous agents d'une classe ont cette variable.
- La valeur de cette variable est individuelle pour chaque agent de la classe.
- Certaines variables sont préexistantes
  - ex: coordonnées

```
patches-own[
    living?
    living-neighbors
]
```

- Conway
  - Patches:
    - Etat, vivant ou mort
    - Nombre de voisins vivants

Procédures

- Les procédures sont les éléments de base du programme Netlogo
- Les procédures sont liées à un agentset (contextes):
  - observer (modèle)
  - patches
  - turtles…
- Les procédures peuvent être appelées par une autre procédure ou par un bouton de l'interface

```
to <nom_de_la_procédure>
    <la procédure>
end
```

Procédures: arguments

- Les procédures peuvent accepter des arguments
- Les arguments sont des variables locales de la procédure qui lui sont transmises lorsqu'elle est appelée

```
to <nom_de_la_procédure> [args]
    <la procédure>
end

to add [x y]
    let z (x + y)
    print z
end
```

Procédures: report

- Une procédure peut retourner une variable.

```
to-report <nom> [args]
    <la procédure>
    report <variable>
end

to-report add [x y]
    let z (x + y)
    report z
end
```

setup et go

- Par convention:
  - setup est la procédure d'initialisation du modèle
  - go est la procédure répétée à chaque pas de temps du modèle

```
to setup
    <initialisation du modèle>
end

to go
    <itération du modèle>
end

to setup
  ask patches [
    set pcolor white
    set living? false
    ]
end
```

Généraliser

- Tout code qui doit être appelé depuis différents contextes devrait être dans une procédure séparée.
- Cette procédure sera le seul endroit où modifier ce code.
- Réduit les risques d'erreurs

```
to setup
  ask patches [
    ;All patches are dead by default
    set-dead
    ]
end

;;PATCHES PROCEDURES;;

to set-dead
  ;Set dead patches variables
  set pcolor white
  set living? false
end
```

Structures de contrôle

- Les structures de contrôle commande si, combien de fois et dans quel ordre les instructions sont exécutées, éventuellement en fonction de conditions.
- Structures:
  - `ask`: itération d'un agentset
  - `if` contrôle si une condition est remplie
  - `while` exécute tant qu'une condition est remplie
  - `loop/stop` boucle infinie
  - `foreach` itère sur une liste

```
to setup
  ask patches [
    ;All patches are dead by default
    set-dead
    ]
end

;;PATCHES PROCEDURES;;

to set-dead
  ;Set dead patches variables
  set pcolor white
  set living? false
end
```

Créer une proportion de cellules vivantes

- Vérifie si random-setup est sur on (booléenne)
- Calcule le nombre de cellules qui doivent être vivantes
- Demande à ces cellules de devenir vivantes

```
to make-random-setup
  ;Create a proportion of living patches
  if random-setup? [
    let to-live (count patches * random-
setup-p)
    ask n-of to-live patches [
      set-living
      ]
    ]
end
```

Netlogo

Cellules

Règles

Règles de transitions

- Trois possibilités
  - Naît
  - Survit
  - Meure
- Dépend du nombre de cellules voisines vivantes à t-1
- Game of Life:
  - Naît si 3 voisines vivantes
  - Survit si 2 ou 3 voisines vivantes
  - Sinon, meure

Listes

- Les règles de transitions seront implémentées dans des listes.
- Les listes sont des variables permettant de stocker plusieurs autres variables.
- Game of Life: on stocke la liste du nombre de voisines vivantes par transition possible:
  - `birthlist [3]`
  - `survivelist [2 3]`

```
globals [
     birthlist
     survivelist
]

…
```

Transition

- Il ne reste plus qu'à vérifier, pour chaque cellule, si elle se trouve dans une des trois transitions possibles à chaque tour.
- Procédure transition

```
to transition
  ifelse member? living-neighbors
birthlist [
     set-living
  ] [
     if not member? living-neighbors
survivelist
    [
          set-dead
      ]
  ]
end
```

If et ifelse

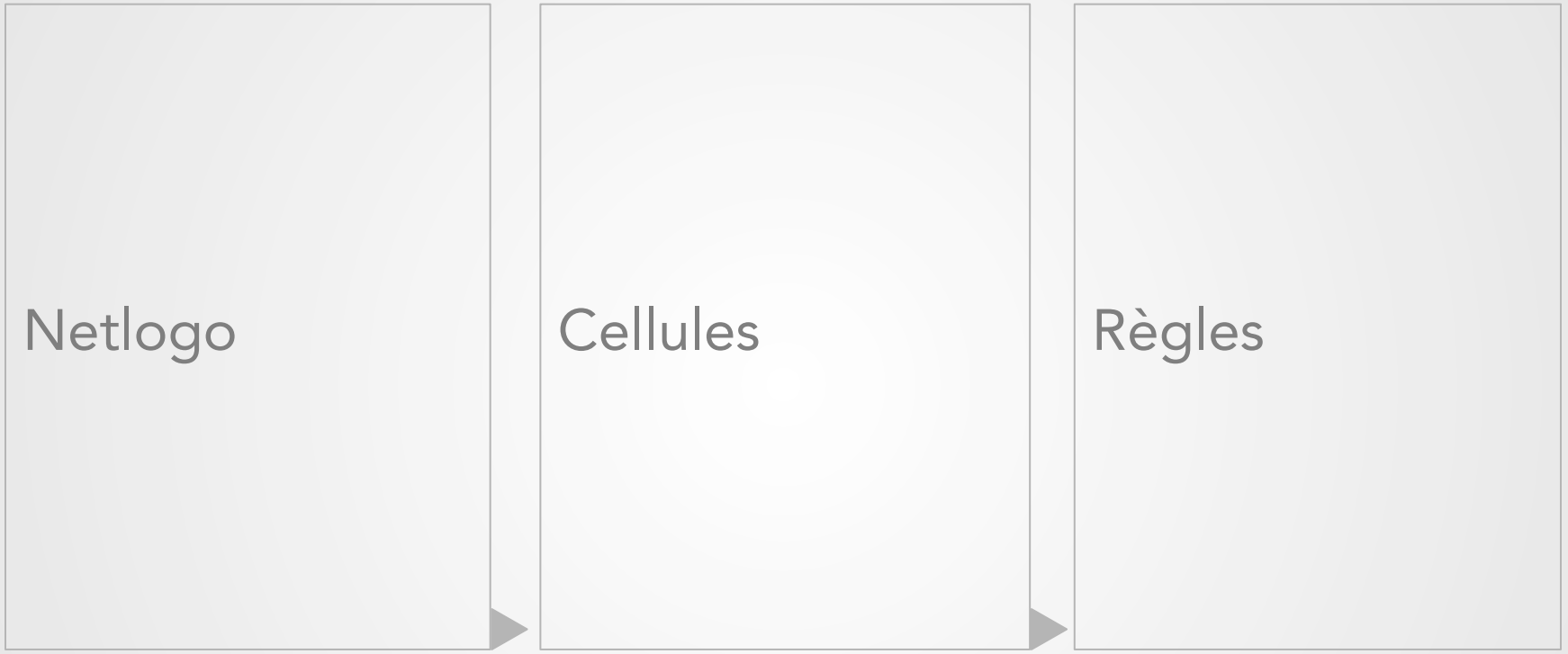
- `if` permet d'exécuter un bloc de code si une condition logique est vérifiée.
- `ifelse` exécute un bloc si la condition est vérifiée, et un autre dans le cas contraire
- `not` permet d'inverser une condition logique

```
to transition
  ifelse member? living-neighbors
birthlist [
     set-living
  ] [
     if not member? living-neighbors
survivelist
    [
          set-dead
      ]
  ]
end
```

Bonus: dessiner les cellules vivantes à la souris

- `while` dessiner tant que le bouton de la souris est enfoncé
- variable `mouse-down?`
- `ifelse` inverser l'état de la cellule

```
to draw-cells
  let erasing? [living?] of patch mouse-
xcor mouse-ycor
  while [mouse-down?]
    [ ask patch mouse-xcor mouse-ycor
      [ ifelse erasing?
        [ set-dead ]
        [ set-living ] ]
      display ]
end
```


Netlogo
Cellules
Règles