Programmation de machines amorphes

Valvassori Moïse

Laboratoire d'Intelligence Artificielle
Université Paris 8

18/07/2007

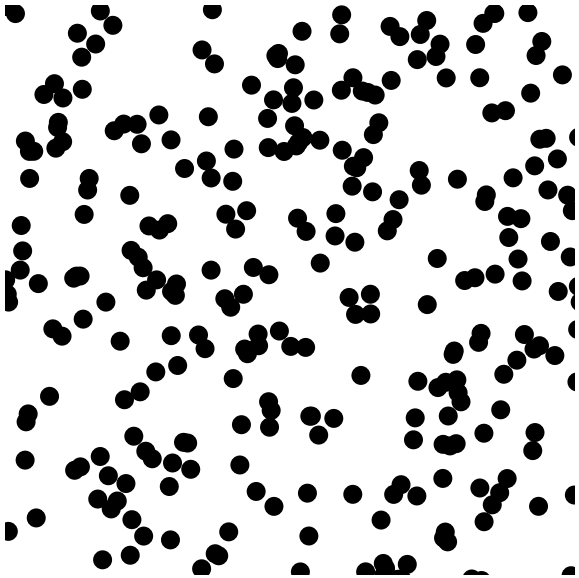
Ordinateur Amorphe

Ordinateur Amorphe :

- Ensemble d'éléments de calcul

où les éléments :

- sont identiques
- sont spatialement distribués
- ne connaissent pas leur position
- asynchrone
- communiquent par broadcast avec leur voisinage



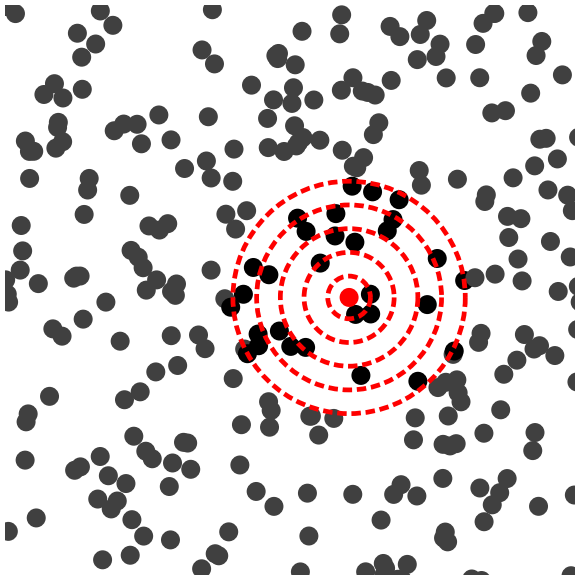
Ordinateur Amorphe

Ordinateur Amorphe :

- Ensemble d'éléments de calcul

où les éléments :

- sont identiques
- sont spatialement distribués
- ne connaissent pas leur position
- asynchrone
- communiquent par broadcast avec leur voisinage



Motivation

Concevoir et implémenter des langages de haut niveau sur des ordinateurs amorphes.

Conception multi-niveaux :

niveau	contenu
niveau 0	élément amorphe
niveau 1	langage bas niveau
niveau 2	primitives bas niveau
niveau 3	primitives haut niveau
niveau 4	langage haut niveau

Motivation

Concevoir et implémenter des langages de haut niveau sur des ordinateurs amorphes.

Conception multi-niveaux :

niveau	contenu	exemples
niveau 0	élément amorphe	senseurs, wifi ad-hoc, simulation
niveau 1	langage bas niveau	tical
niveau 2	primitives bas niveau	gradient, déplacement des états
niveau 3	primitives haut niveau	Blob
niveau 4	langage haut niveau	BlobML like, ...

Motivation

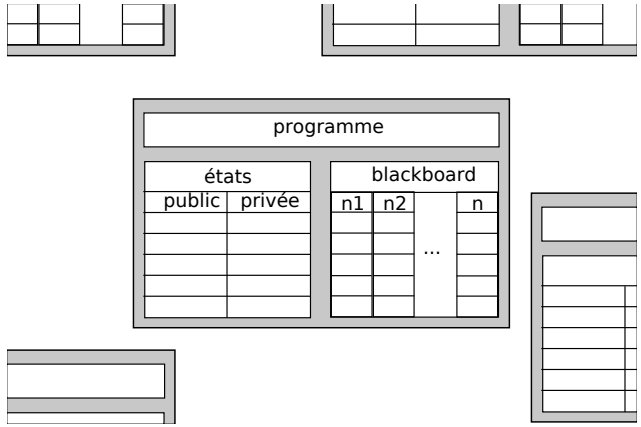
Concevoir et implémenter des langages de haut niveau sur des ordinateurs amorphes.

Conception multi-niveaux :

niveau	contenu	exemples
niveau 0	élément amorphe	senseurs, wifi ad-hoc, simulation
niveau 1	langage bas niveau	tical
niveau 2	primitives bas niveau	gradient, déplacement des états
niveau 3	primitives haut niveau	Blob
niveau 4	langage haut niveau	BlobML like, ...

Modèle d'éléments amorphes

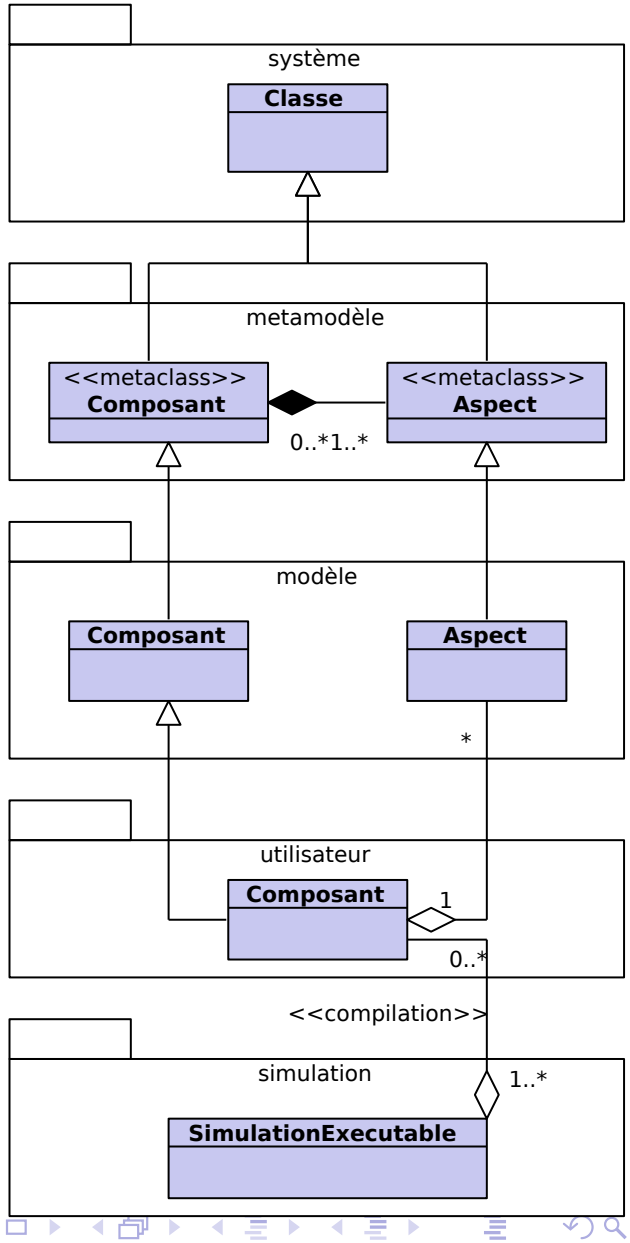
- états :
  - privés
  - publics $\longrightarrow$ messages
- tableau noir contenant les états publics du voisinage
- programme
- communication parfaite



Tical : Langage de Programmation Amorphe

*T*ical *i*s a *C*omponent and *A*spect *L*anguage
*T*ical *i*s a *C* *A*morphous *L*ibrary

- compile vers simulation en C.



Aspects

- séparation des intérêts
- tisseur d'aspects
- point de jonction : label

Aspects : Exemple

```
(declare-aspect expl1 #:pointcut 'jp1)
  => <aspect-expl1>
(define-method (compile (a <aspect-expl1>))
  (do-thing))
(define-method (compile (a <aspect-expl1>))
  (do-other-things))
```

Aspects : Exemple

```
(declare-aspect expl1 #:pointcut 'jp1)
  => <aspect-expl1>
(define-method (compile (a <aspect-expl1>))
  (do-thing))
(define-method (compile (a <aspect-expl1>))
  (do-other-things))

;; ailleurs dans le programme :
(interesting-code)
(compile (make-label-joinpoint 'jp1))
(continuing-code)
```

Aspects : Exemple

```
(declare-aspect expl1 #:pointcut 'jp1)
  => <aspect-expl1>
(define-method (compile (a <aspect-expl1>))
  (do-thing))
(define-method (compile (a <aspect-expl1>))
  (do-other-things))

;; ailleurs dans le programme :
(interesting-code)
(compile (make-label-joinpoint 'jp1))
(continuing-code)

;; Programme effectivement exécuté :
(interesting-code)
(do-thing)
(do-other-things)
(continuing-code)
```

Aspects pour la programmation amorphe

- `#:states` : déclaration des états privés
- `#:messages` : déclaration des états publics
- `#:init` : initialisation des éléments
- `#:message-handlers` : traitements des messages
- `#:behaviours` : comportements généraux

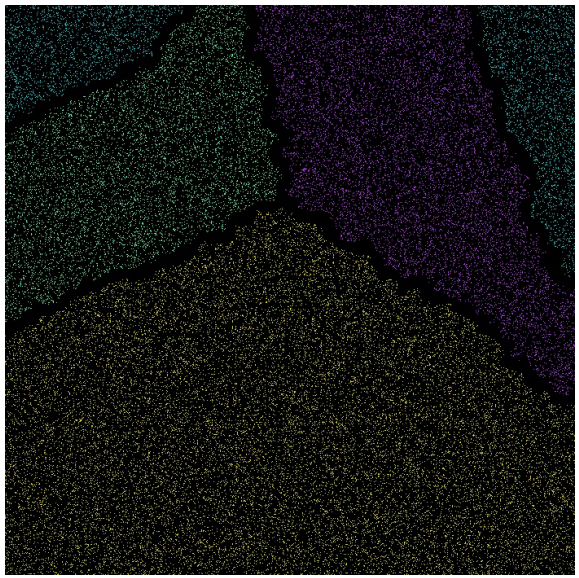
Composants

- Construit des abstractions
- Un composant utilise d'autres composants
- Un utilisateur a accès aux états des utilisés
- Deux nouveaux aspects :
  - `#:uses` : utilisation de composants
  - `#:param` : déclaration de paramètres pour le composant
- assemblage d'un «aspect effectif» à la compilation

Exemple : Voronoï

Construire un diagramme de Voronoï.
Marquer les éléments d'un des deux états suivants : *empty* ou *line*
Algorithme :

- initilisation :
  - initialiser les valeurs de propagation
  - choisir 5 éléments et mettre leur valeur de propagation à *random*
- propager les valeurs de proche en proche
- si un élément reçoit deux valeurs différentes alors il est sur une ligne



Exemple : Voronoï

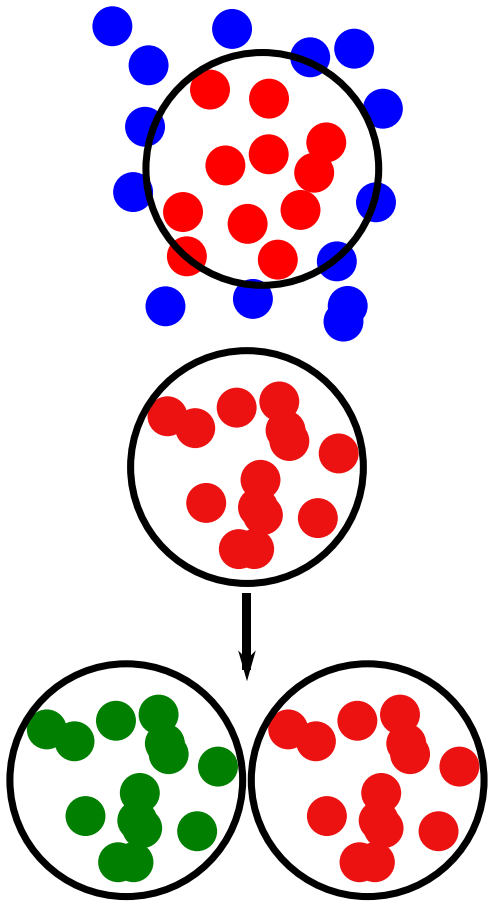
```
(make-component voronoi
  #:uses ((p <propagation>))
  #:states ((state (empty line)))
  #:init ((set! s:state empty))
  #:init:after ((with-n-cells-do 5 (set! s:p_value (rand))))
  #:message-handlers ((p_propage (if (!= m:p_propage s:p_value)
                                     (set! s:state line)))))
```

Exemple : Voronoï

```
(make-component voronoi
  #:uses ((p <propagation>))
  #:states ((state (empty line)))
  #:init ((set! s:state empty))
  #:init:after ((with-n-cells-do 5 (set! s:p_value (rand))))
  #:message-handlers ((p_propage (if ( != m:p_propage s:p_value)
                                     (set! s:state line)))))
(make-component propagation
  #:params ((init 0)                    (propagation-method s:value))
  #:states (value)
  #:messages (propage)
  #:init ((set! s:value p:init))
  #:message-handlers ((propage
             (when-and (( != m:propage p:init)  (= s:value p:init))
                       (set! s:value m:propage))))
  #:behaviours ((if ( != s:value p:init)
                  (set! m:propage p:propagation-method)
                  (set! m:propage p:init))))
```

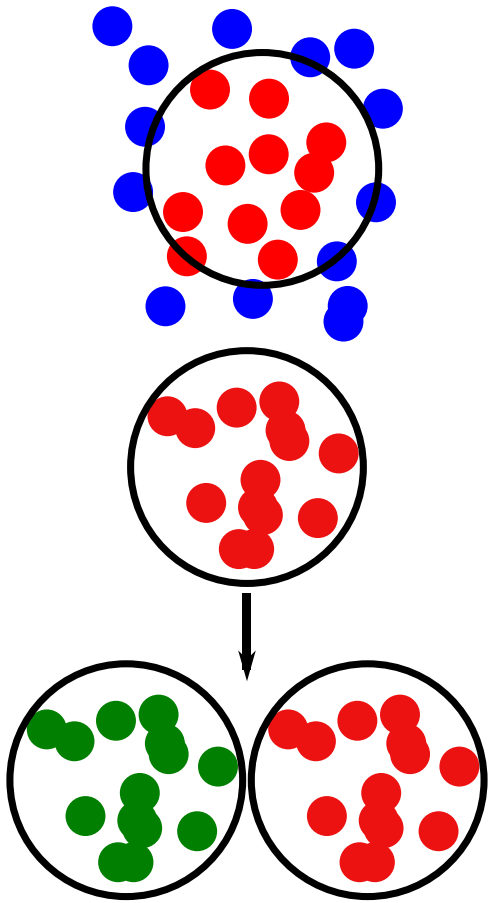
Blob [Gruau, 2002]

- Blob est un ensemble d'éléments amorphe
- Blob peut se déplacer (déplacement des états des éléments)
- Primitives Blob :
  - construire un Blob
  - détruire un blob Blob
  - diviser un Blob
  - fusionner deux Blobs
  - lier deux Blobs
- implémentation inspirée de la biologie



Blob [Gruau, 2002]

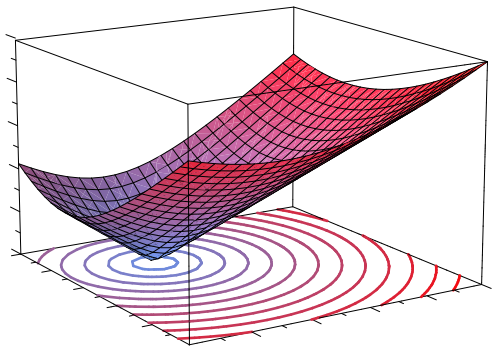
- Blob est un ensemble d'éléments amorphe
- Blob peut se déplacer (déplacement des états des éléments)
- Primitives Blob :
  - construire un Blob
  - détruire un blob Blob
  - diviser un Blob
  - fusionner deux Blobs
  - lier deux Blobs
- implémentation inspirée de la biologie



Construire un Blob

Inspiré de la Stigmerie
Algorithme (//) :

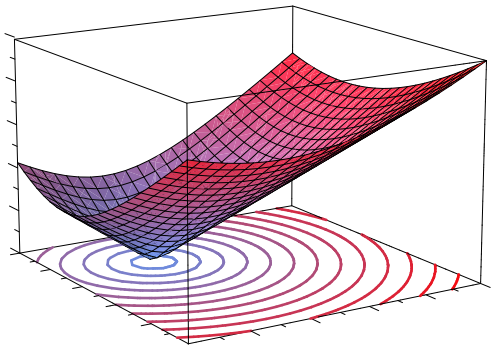
- un élément du Blob peut initier un gradient
- les éléments du Blob descendent les gradients
- les gradients s'évaporent

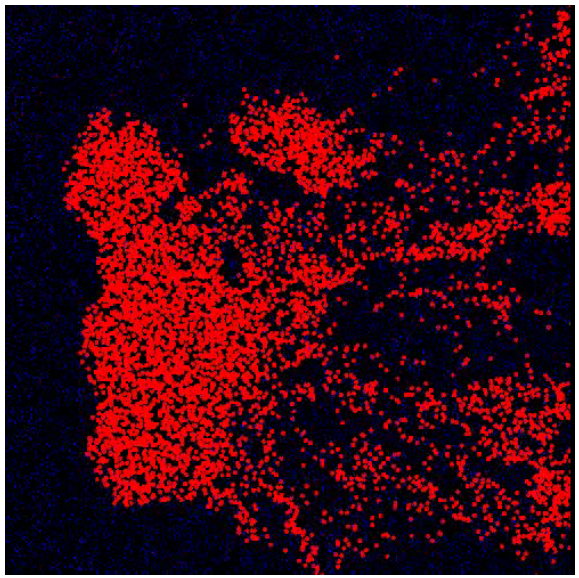


Construire un Blob

Inspiré de la Stigmerie
Algorithme (//) :

- un élément du Blob peut initier un gradient
- les éléments du Blob descendent les gradients
- les gradients s'évaporent

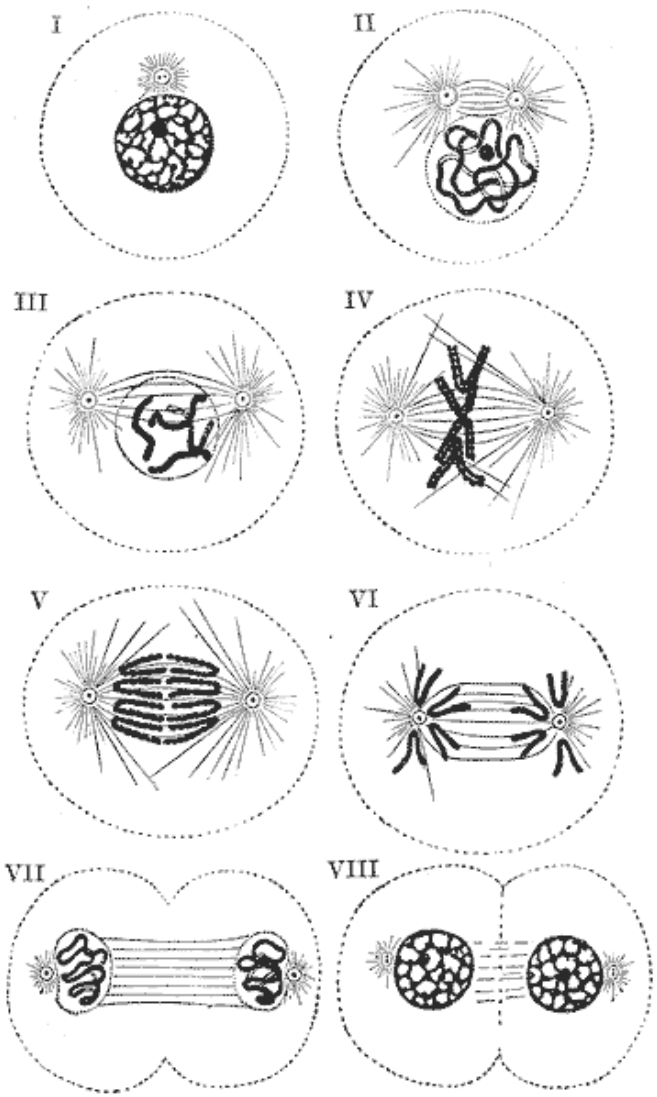




Division d'un Blob : Modèle

Inspirée par la mitose des cellules eucaryotes
Phases de la mitose :

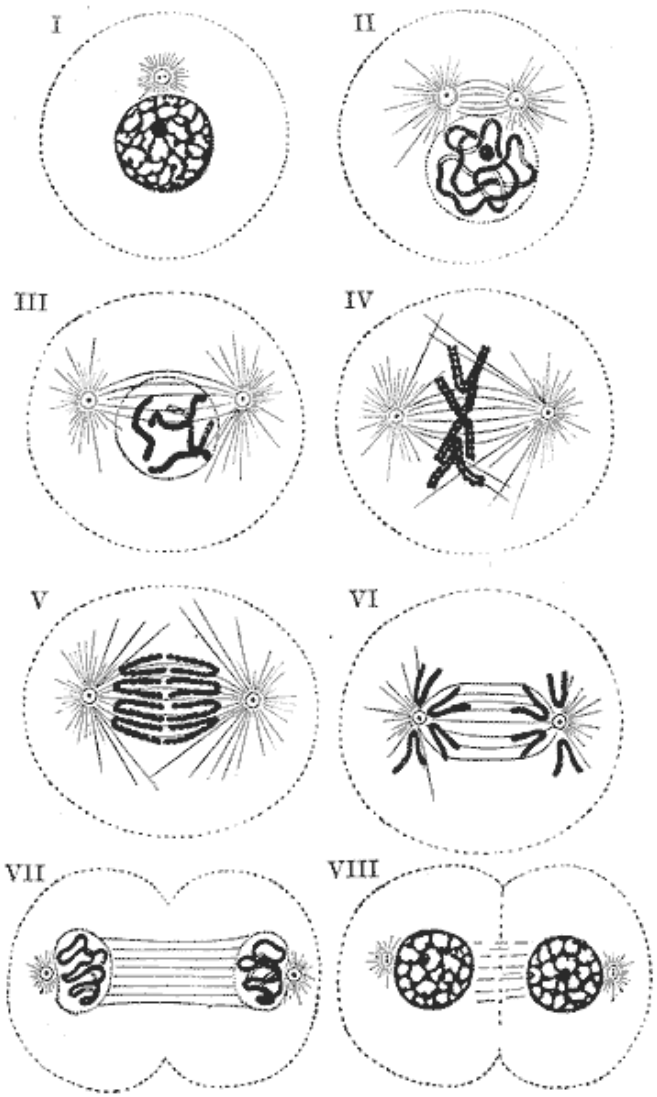
- Interphase (G1 - G2)
- Prophase (I - III)
- metaphase (IV)
- anaphase (V - VI)
- Telophase (VII - VII)



Division d'un Blob : Modèle

Inspirée par la mitose des cellules eucaryotes
Phases de la mitose :

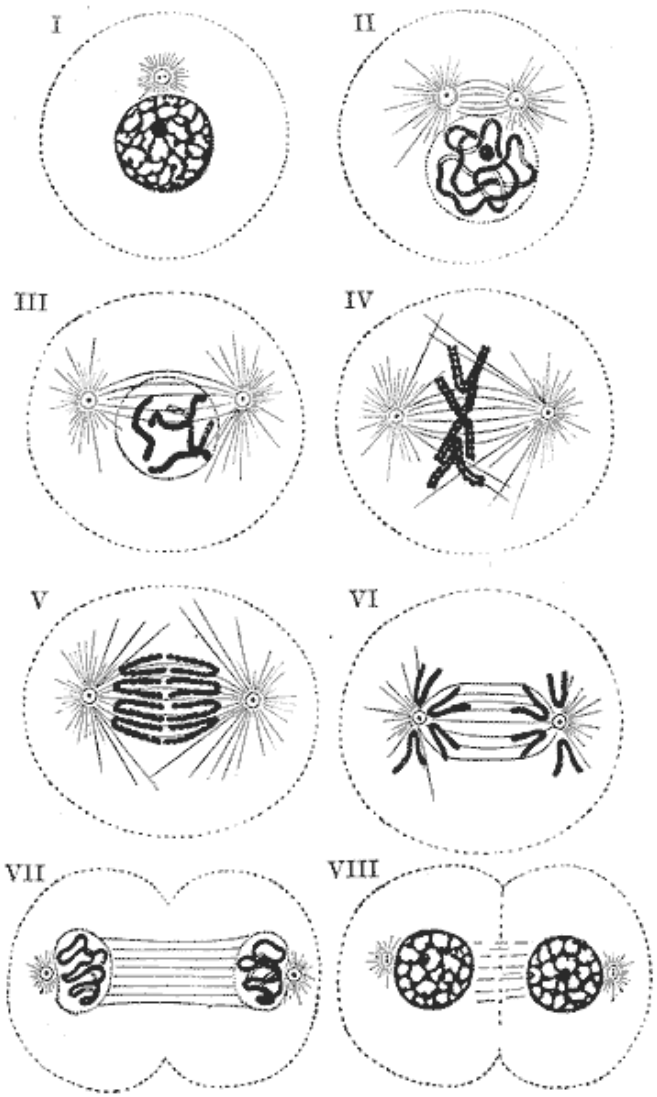
- Interphase (G1 - G2)
- Prophase (I - III)
- metaphase (IV)
- anaphase (V - VI)
- Telophase (VII - VII)



Division d'un Blob : Modèle

Inspirée par la mitose des cellules eucaryotes
Phases de la mitose :

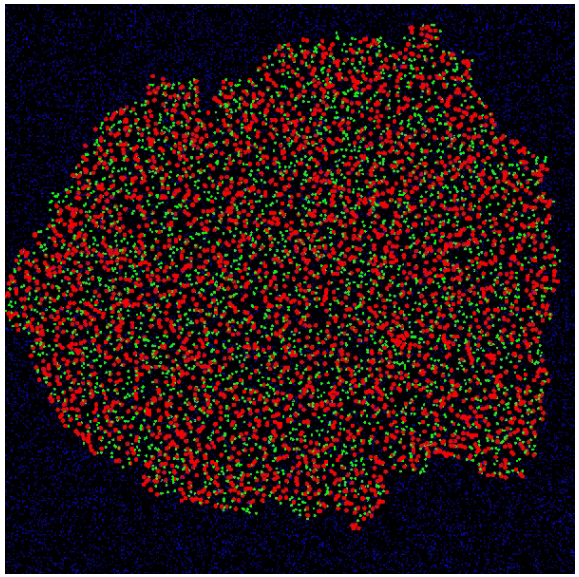
- Interphase (G1 - G2)
- Prophase (I - III)
- metaphase (IV)
- anaphase (V - VI)
- Telophase (VII - VII)



Division des Blobs : Interphase (G1)

Algorithme (//) :

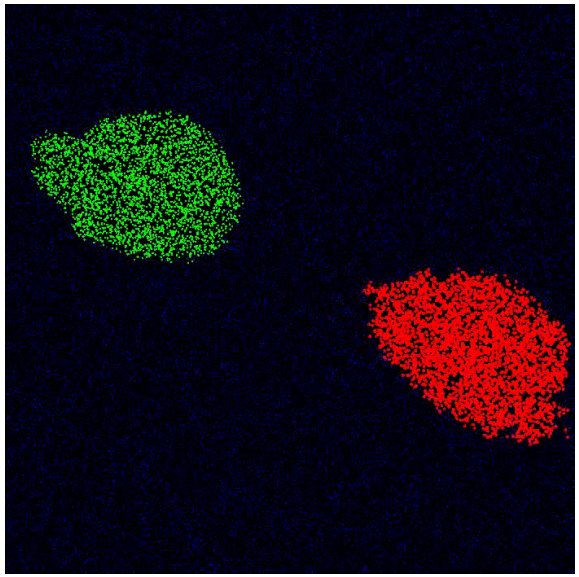
- duplication si $\neg$ déjà dupliqué $\wedge$ élément vide est trouvé
- les éléments Blob peuvent initié un gradient
- les éléments Blob remontent les gradients si la densité des éléments Blob $> d$
- les gradients s'évaporent



Blob Division : Prophase+Telophase

Dans la même phase

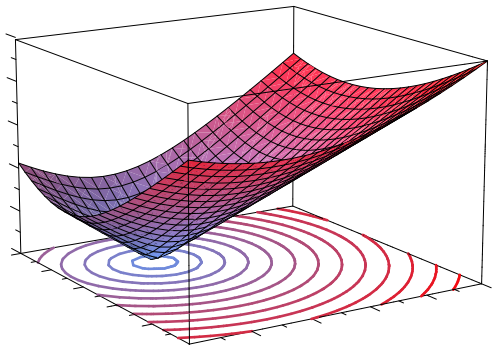
- Prophase : brisure de symétrie
- Telophase : séparation des Blobs

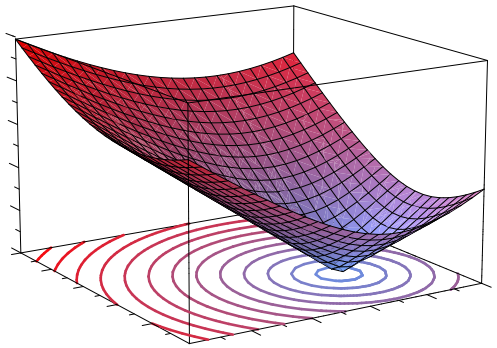


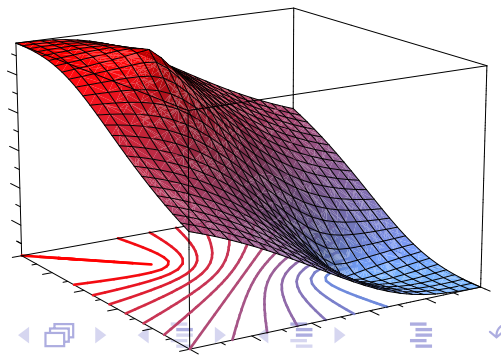
Blob Division : Prophase+Telophase

Algorithme (//) :

- Prophase
  - élire un leader
  - les leaders initient un gradient
  - soustraire le gradient *leader* avec le gradient *leader* de l'autre blob $\longrightarrow G$
- Telophase
  - les éléments Blob sur le minima de G peuvent initier un gradient de recrutement
  - construire le blob

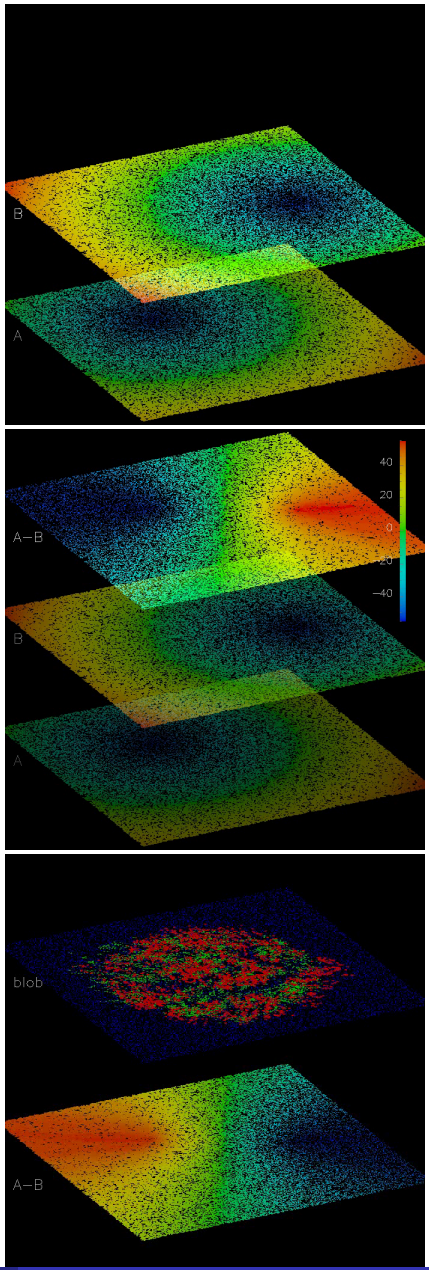






Blob Division : Prophase+Telophase

- Élection des leaders
- Construction de G
- Gradient G et les Blobs



Conclusion

- méthodologie générale
- langage bas niveau dédié
- implémentation de primitives Blob
  - inspirées de la biologie

Perspectives

- Compléter les primitives Blobs (communication)
  - faire fonctionner les primitives ensembles
    - passage à l'échelle
    - adressage
  - contrôle temporel des Blobs : FSM
- Implémenter le langage haut niveau
  - Blob Abstract Machine
  - BlobML

- explorer des niveaux alternatifs : changer de langage haut, bas niveau, de modèle d'éléments
- problème de la croissance exponentielle

«

- algèbre des gradients» (?)

(interlude)

Différence de deux gradients