Annexe : exemple d'application

Cette annexe présente une illustration du modèle acteur-observateur réalisée en SmalltalkAgents (voir paragraphe 4.2 page 67). Nous montrons comment mettre en oeuvre un système d'animation par modèle physique puis par modèle d'évolution au moyen d'observateurs. Nous montrons alors que la notion de méta-observateur permet de combiner de façon harmonieuse ces deux modèles d'animation.

Nous avons pris pour exemple un système consititué d'un cube et d'une base de temps. Dans le cas de l'animation par modèle physique, le cube est soumis à une force de gravité. Dans le cas de l'animation par modèle d'évolution, le cube se duplique pour former une chaîne de cubes.

Dynamique newtonienne

Pour créer une animation par dynamique newtonienne (voir paragraphe 6.1.3 page 108), on commence par créer les acteurs suivants :

- *box* : cube affiché dans une fenêtre 3D;
- *time* : base de temps. Reçoit un message *tick* à intervalles réguliers;
- *pos* : position du cube;

- *speed* : vitesse du cube;
- *resultant* : force appliquée au cube.

FIGURE 56 Création des acteurs de l'animation (code commun à tous les exemples)

```
animateDynamics

      ß Directives Hidden ß

      ≥Open our first window≤
      window := (AnimationWindow new: self)
            label: 'Animation';
            open.

      ≥Create a visible box≤
      box := BoxNode new initialize; active.
      (window@#'3DPane') addObject:
            (box box value). "force immediate value"

      ≥Create a time base≤
      time := TimeBase new initialize; active.
      time
            setMasterClock: clock slaveZero: (clock getTime);
            setStartTime: (clock getTime);
            setRate: 65536;
            period: 1000;
            "tick"
            start.

      ≥Create an active position≤
      pos := Position new
            position: ((Quickdraw3DLibrary@#Point3D)
                        new x: -2.0; y: -4.0; z: -4.0);
            scale: 1.0.
      box position: pos.
      pos := pos active.

      ≥Create an active speed≤
      speed := Speed new
            speed: ((Quickdraw3DLibrary@#Vector3D)
                        new x: 0.2; y: 1.5; z: 0.2);
            growth: 1.0.
      speed := speed active.

      ≥Create an active resultant≤
      resultant := Resultant new
            resultant: ((Quickdraw3DLibrary@#Vector3D)
                        new x: 0.0; y: -0.2; z: 0.0);
            pression: 1.0.
      resultant := resultant active.
```

On crée alors un observateur de dynamique *dyna* faisant interagir *time*, *pos*, *speed* et *resultant*. L'envoi du message *start* à *dyna* place un message sémantique sur *time* permettant à *dyna* de recevoir un message *tick* chaque fois que *time* le reçoit et d'établir ainsi une interaction. Lors de la réception d'un message *tick*, *dyna* modifie la vitesse et la position de la boîte conformément aux lois de la dynamique.

FIGURE 57 Création et initialisation d'un observateur de dynamique

```
≥Create an active dynamics and animate≤
dyna := Dynamics new active.
dyna
      timeBase: time;
      speed: speed;
      position: pos;
      resultant: resultant;
      weight: wg;
      dt: 1.0;
      start.
```

FIGURE 58 Initialisation de l'interaction

```
start

      ß Directives Hidden ß

      tb
            on: #tick send: #tick to: self.
```

FIGURE 59 Implantation de l'interaction dynamique

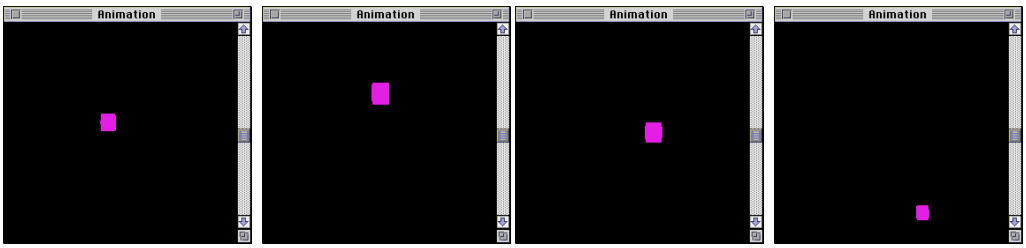
```
tick

      | temp |

    ß Directives Hidden ß

      temp := (Quickdraw3DLibrary@#Vector3D) new.
      (r resultant) scale:
            ((Quickdraw3DLibrary@#Float) new value: dt) result: temp.
      s move: temp.
      s grow: (r pression value).
      (s speed) scale:
            ((Quickdraw3DLibrary@#Float) new value: dt) result: temp.
      p move: temp.
      p grow: (s growth value).
```

FIGURE 60 Séquence d'animation (dynamique)



Modèle d'évolution

Pour créer une animation par modèle d'évolution (voir paragraphe 6.2.1 page 111), on commence par créer les acteurs suivants :

- *box* : cube affiché dans une fenêtre 3D;
- *time* : base de temps. Reçoit un message tick régulièrement envoyé par le gestionnaire multimédia Quicktime de MacOS;
- *pos* : position du cube;
- *orientation* : orientation de la croissance;

On crée alors un observateur d'évolution *evol* faisant interagir *time*, *pos* et *orientation*. L'envoi du message *start* à *evol* place un message sémantique sur *time* permettant à *dyna* de recevoir un message *tick* chaque fois que *time* le reçoit et d'établir ainsi une interaction. Lors de la réception d'un message *tick*, *evol* fait croître une valeur d'énergie qui, lorsqu'elle dépasse un certain seuil, conduit à l'envoi du message *replicate* et à la duplication de *box*.

FIGURE 61 Création et initialisation d'un observateur d'évolution

```
≥Create an active evolution and animate≤
evol := Evolution new active.
evol
      timeBase: time;
      origin: pos;
      replication: 4.0;
      energy: 0.0;
      sons: {};
      deviation: 0.2;
      orientation: orientation;
      view: (window@#'3DPane');
      start.
```

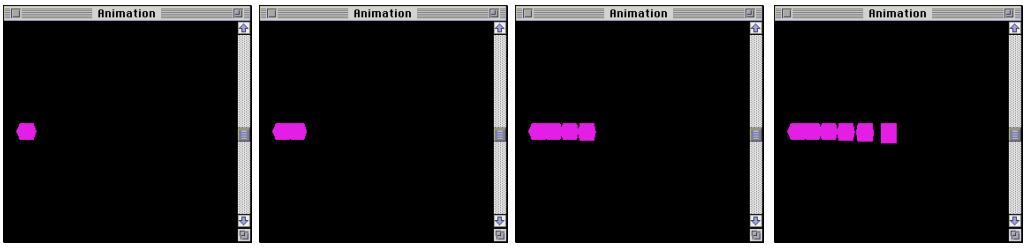
FIGURE 62 Implantation de l'évolution

```
tick

      ß Directives Hidden ß

      energy := energy + 1.0.
      (energy > replication) ifTrue: [
            energy := 0.0.
            done ifFalse: [done:=true. self smPerform: #replicate.].
            done:=true.].
```

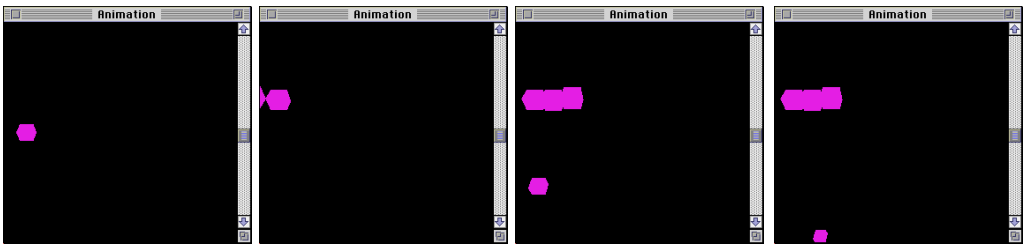
FIGURE 63 Séquence d'animation (évolution)



Combinaison de la dynamique et de l'évolution

Afin de combiner au sein d'une même animation modèles physiques et évolutifs (voir paragraphe 6.2 page 110), on peut tenter de créer tous les acteurs et observateurs décrits ci-dessus. Le résultat est en effet une combinaison décevante : la boîte tombe et pousse mais les boîtes créées ne tombent pas sous l'effet de la gravité.

FIGURE 64 Séquence d'animation (mauvaise combinaison)



Afin de combiner harmonieusement dynamique et évolution, on crée un méta-observateur *meta* pour modifier le comportement d'*evol*. Tout envoi du message *replicate* par *evol* est transmis à *meta* grâce à un message sémantique défini dans la méthode *start* de *meta*. Lors de la réception d'un message *replicate* par *meta*, la dynamique appliquée à l'acteur que fait interagir *evol* est dupliquée et appliquée au nouvel acteur créé par *evol*.

Ainsi, la boîte tombe, « pousse » et les boîtes créées tombent.

FIGURE 65 Création et initialisation d'une métaobservation

```
≥Make evolution and dynamics interact≤
 meta:= DynEvol new active
      evol: evol;
      start.
```

FIGURE 66 Initialisation de l'interaction

```
start

      ß Directives Hidden ß

      evol
            on: #replicate send: #replicate to: evol;
            on: #replicate send: #replicate to: self.
```

FIGURE 67 Implantation de la combinaison harmonieuse

```
replicate

      | actor dyna0 |

      ß Directives Hidden ß

      actor := evol origin.
      dyna0 := (Dynamics for: actor) at: 1.

      (evol sons) do: [:s |
            | speed resultant wg dyna evoli meta |
            ≥Create an active speed≤
            speed := dyna0 speed copy.
            ≥Create an active resultant≤
            resultant := dyna0 resultant copy.
            ≥Create a passive weight≤
            wg := dyna0 weight copy.
            ≥Create an active dynamics and animate≤
            dyna := Dynamics new active.
            dyna
                  timeBase: (dyna0 timeBase);
                  speed: speed;
                  position: s;
                  resultant: resultant;
                  weight: wg;
                  dt: 1.0;
                  start.

            evoli := (Evolution for: s) at: 1.
            meta := DynEvol new active
                  evol: evoli;
                  start.
      ]
```

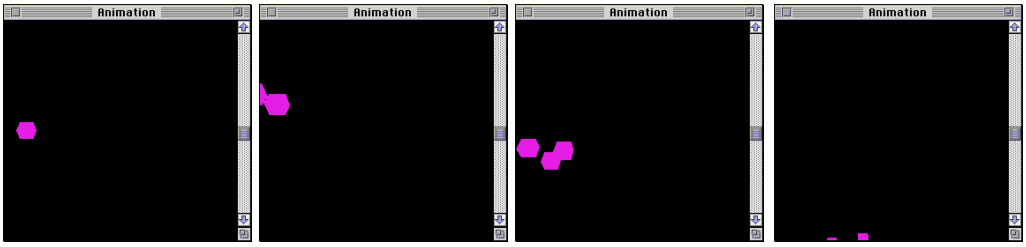



FIGURE 68 Séquence d'animation (combinaison harmonieuse)