MINES Douai

Concurrence

Noury Bouraqadi

1 Thread et affichage textuel

Vous allez utiliser le Transcript dans cet exercice et analyser le texte qui y apparaitra. Afin de vous simplifiez la vie, effacez son contenu avant chaque exécution. Il est possible de faire cela directement dans votre code à l'aide de l'expression `Transcript clear`.

1.1 Threads simples

Définissez deux threads en faisant appel à la méthode `fork` et qui tous les deux effectuent un affichage le `Transcript`.

— le premier affiche 10 fois la chaîne de caractères `'ping'` précédée d'un saut de ligne ;
— le second affiche 10 fois la chaîne de caractères `'PONG'` précédée d'un saut de ligne.

Quel affichage obtenez-vous ? Expliquez.

1.2 Priorités de threads

Modifiez vos threads de manière à leur attribuer des priorités différentes :
— attribuez la priorité 30 au premier thread (qui affiche 'ping') ;
— attribuez la priorité 40 au second thread (qui affiche 'PONG')

Quel affichage obtenez-vous ? Expliquez.

1.3 Attentes

Ajoutés des délais d'attentes dans vos threads :
— attendez 100 millisecondes avant chaque affichage pour le premier thread (qui affiche 'ping') ;
— attendez 300 millisecondes avant chaque affichage pour le second thread (qui affiche 'PONG').

Quel affichage obtenez-vous ? Expliquez.

2 Balles Rebondissantes

Soit la classe `SimulationBalleRebondissante` fournie qui correspond à une simulation de balle rebondissante. Elle met en oeuvre un *thread* qui contrôle le mouvement d'une balle. Elle répond aux méthodes :

— `lancerSimulation` affiche une balle rebondissante et lance le thread qui en contrôle le déplacement.
— `arreterSimulation` arrête le thread qui contrôle le déplacement de la balle.
— `arreterAffichage` masque l'affichage de la balle rebondissante.

1. Testez le bon fonctionnement d'une simulation.
2. Lancez une simulation et sauvegardez votre image et Pharo **sans** quitter la simulation. Relancez votre image Pharo. Que constatez-vous
3. Construire, puis lancez simultanement une collection de 100 simulations. Que constatez-vous ?
4. Proposez une solution qui résout le problème.

3 Course d'escargots

Vous allez réaliser dans cet exercice une course d'escargots. Les escargots seront seront affichés sous forme de graphique à l'aide de la classe `AfficheurEscargot fournie`. Vous utiliserez le *pattern* observateur pour actualiser l'affichage.

La course est chronométrée. Votre application doit avoir un thread pour le chronometre, ainsi qu'un thread par escargot. Ainsi, une course de 3 escargot nécessitera 4 threads. Lorsque le premier escargot a depassé la ligne d'arrivé, tous les threads doivent être arrêtés et la valeur du chronomètre doit être affichée sur le `Transcript`.

Les escargots sont initialement alignés sur la gauche de l'écran et doivent parcourir une distance equivalente à plusieurs fois leurs longeur (un escargot fait 150 pixels de long). Un escargot avance d'un pas (10 pixels) à chaque pas de temps (100 millisecondes). Il dispose d'une quantité d'énergie qui correspond au nombre de pas consécutifs qu'il peut effectuer. A chaque pas, le niveau d'énergie est décrémenté.

Lorsque le niveau d'énergie atteint 0 l'escargot rentre dans sa coquille et dors pendant un nombre de pas de temps aléatoire (entre 1 et 10). Pendant le sommeil, le niveau d'énergie est incrémenté d'une valeur aléatoire entre 1 et 10. Lorsque l'escargot se réveille, il reprend la course.

A sa création, la quantité d'energie initiale d'un escargot est définie de manière aléatoire entre 1 et 10. Pour le tirage aléatoire, vous utiliserez la méthode `atRandom` permet d'obtenir un élément au hasard d'une collection. Par exemple l'expression `(25 to: 43) atRandom` retourne un élément au hasard dans l'intervalle de nombres entiers dont les bornes sont 25 et 43 inclus.