

TD7

État

Nous proposons de réaliser un digicode qui vérifie que la séquence "a b c" a été entrée sur un clavier composé de 4 touches : a, b, c et d. La figure 1 illustre la machine à états de ce digicode.

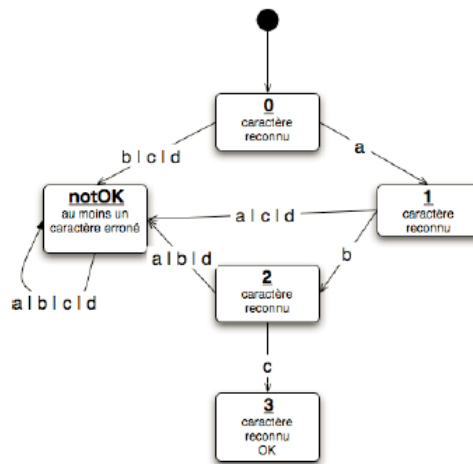


FIGURE 1 – Machine à états du digicode

1. Écrivez une classe `Digicode` contenant une méthode `appuyer` qui prend en paramètre un caractère (*i.e.*, la touche appuyée) et, en fonction de ce dernier, change l'état courant du digicode, en utilisant des constantes pour représenter les différents états de l'automate.
2. Quels sont les inconvénients de cette solution, en particulier par rapport à la modification de la machine à états du digicode (*e.g.*, si la séquence à vérifier devient "a c d c") ?
3. Proposez un diagramme de classes qui modélise le problème en utilisant le pattern *État*.
4. Écrivez une implémentation des classes du diagramme. Lorsque la transition vers l'état 3 est prise, un message indiquant l'ouverture de la porte d'entrée est affiché dans la console.
5. Écrivez un programme de test qui demande à l'utilisateur de composer un code.
6. Changez le code du digicode (*i.e.*, sa machine à états) par la séquence "a b b a".

Stratégie

Nous proposons de réaliser le jeu *pierre-feuille-ciseaux*. Dans ce jeu, deux joueurs s'opposent à coups de *pierre*, *feuille* et *ciseaux*. La *pierre* bat les *ciseaux* (en les émoussant), les *ciseaux* battent la *feuille* (en la coupant), la *feuille* bat la *pierre* (en l'enveloppant). Si les deux joueurs jouent le même coup, il y a égalité.

1. Écrivez une classe `Joueur` qui représente un joueur (*i.e.*, son nom et son score). Cette classe contient notamment une méthode `ajouterUnPoint()` qui ajoute un point au score

- du joueur, et une méthode `jouer()` qui retourne le coup choisi par le joueur selon sa stratégie. Une classe `Attack.java` modélisant un coup est fournie.
2. Proposez un diagramme de classes qui modélise le problème en utilisant le pattern *Stratégie*.
 3. Écrivez une implémentation des classes du diagramme.
 4. Réalisez trois stratégies :
 - (a) une stratégie qui joue une attaque aléatoirement ;
 - (b) une stratégie qui joue systématiquement la même attaque (fixée à la création) ;
 - (c) une stratégie qui demande à l'utilisateur de choisir une attaque.
 5. Écrivez un programme de test qui crée des joueurs avec différentes stratégies et les oppose pendant un nombre de manches donné. *In fine*, le score des deux joueurs est affiché dans la console.
 6. Expliquez en quoi l'interface `java.awt.LayoutManager` et ses différentes implémentations, telles que `java.awt.BorderLayout` et `java.awt.FlowLayout`, correspondent à un pattern *Stratégie* ?