Licence 3 Info **Corrigé**

LI5a : Développement de programmes

(A. Slissenko)

1.
(1a). Expliquez brièvement à quoi sert la spécification des requis, comment elle peut être décrite et comment elle peut être testée.

Réponse. Voilà une réponse possible. Une spécification des requis décrit les propriétés de logiciel à assurées sans dire comment faire ça. Plus formellement, ce sont des propriété des runs du programme qui doit être écrit. Ces propriétés doivent être décrites en termes des entrées et des sorties. Pour tester les requis on construit des runs qui représentent des comportements 'essentiels' du programme (en prenant en compte les critères de construction de jeux de tests) et on teste si ces runs vérifient les propriétés.

(1b). Quel est le but de la première spécification de programme (de "haut niveau") et quel est le rapport de cette spécification avec les requis ? Quels langages peut-on utiliser pour décrire une telle spécification de programme ?

Réponse. Cette spécification doit représenter les traits décrits dans les requis sans aller dans les détails d'implémentation. Quand même, elle doit être assez précise pour qu'on puisse vérifier si cette spécification est conforme aux requis. Cette spécification peut être décrite en langages assez expressifs avec une sémantique bien précise comme, par exemple, ASM ou même Java, Ada etc.

(1c). Décrivez le rôle des diagrammes de cas d'utilisation, des diagrammes de classes et des diagrammes de machine à états ("statechart") dans le processus de développement de logiciel dans le cadre d'UML.

Réponse. La méthodologie d'application de l'UML, ébauchée dans la spécification de ce langage de diagrammes, propose d'utiliser les diagrammes de cas d'utilisation pour décrire l'interface entre le logiciel et l'environnement de ce logiciel, et de plus la structure de comportement du logiciel en gros. L'environnement (cela peut être un autre programme) est représenté par les acteurs (bonshommes) et la structure de comportement du programme est représentée par les cas d'utilisation. Les diagrammes de classes décrivent plutôt la structure du vocabulaire, en particulier, la structure des sortes abstraites, des relations entre eux, leur attributs et transformations. En fin, les diagrammes de machines à états décrivent le comportement du programme. Toutes ces description sont faites à certain niveau d'abstraction qui devient plus en plus détaillé (via des raffinements) au cours de développement du logiciel.

2.
A partir de la spécification décrite ci-dessous donnez un diagramme de cas d'utilisation, un diagramme de classes et un diagramme statechart qui décrivent des "faces différentes" d'un Contrôleur qui gère les jeux de roulette. Donnez aussi les propriétés du Contrôleur (en français) qui représentent la spécification des requis.

La roulette (française) consiste en un plateau (situé dans une cuvette) et en un tableau des mises. Le plateau est divisé en secteurs numérotés par 0,1,...,36 qui sont colorés en rouge et noir (mais pas tous). Le tableau (de roulette française) possède les champs 0,1,...,36, passe (qui correspond aux secteurs 19–36), manque (1–18), pair, impair, noir et rouge. Les mises sont faites sur les champs du tableau. Il y a une bille qui se déplace sur le plateau si le plateau tourne. Au moment de l'arrêt du plateau la bille se trouve sur un seul numéro. Et ce numéro définit les gains. On ne va pas dans les détails du calcul de gains en fonction de ce numéro et des champs avec les mises ; ci-dessous on introduit une fonction abstraite G qui, en particulier, fait ce calcul.

Quand on parle de gains, on suppose qu'ils sont représentés par des entiers, et un entier négatif signifie une perte.

La roulette se trouve dans un casino. Chaque client qui entre dans le casino présente sa pièce d'identité, et les informations concernant son nom, prénom et numéro de sécurité sociale deviennent disponibles au Contrôleur. Chaque client enregistré reçoit une carte permanente qui contient son numéro individuel attribué par le casino et sur laquelle le casino ajoute les sommes payées par le client au casino pour jouer (un analogue des jetons utilisés dans les casinos réels) et ses gains.

Pour jouer un client doit prendre une place à la table de roulette ; on ne discute pas comment les places sont attribuées. Le nombre de place est $r > 5$. Un client joueur dispose d'un clavier avec un dispositif pour entrer sa carte. Le Contrôleur commence chaque séance de jeu en affichant le signal $startMises$. Ensuite il attend le temps Δ pendant lequel les clients font leurs mises. Ensuite le Contrôleur affiche $finMises$.

Après ce signal, avec un délai δ, le Contrôleur bloque les claviers des joueurs et en même temps il commence à vérifier si la somme des mises pour chaque joueur ne dépasse pas la somme indiquée sur sa carte. Une mise totale zéro pour un joueur implique une amende α qui est immédiatement soustraite de sa carte. La valeur de la somme indiquée sur la carte du joueur situé en place J, $1 \leq J \leq r$, est accessible au Contrôleur comme $somme(J)$ et ses mises sont représentées par les valeurs $mise(J, i)$, où i est le numéro de champ. Si la place J est libre, alors $somme(J) = undef$.

Si la somme des mises d'un joueur est plus grande que la somme figurant sur sa carte toute la somme figurant sur sa carte est perdue pour le joueur, et elle est donc comptée comme gain du casino. Sinon la somme des mises est soustraite de sa carte. En même temps le Contrôleur lance le plateau. Lorsque le plateau s'arrête le Contrôleur reçoit le signal *stop*. A ce moment il calcule les gains en utilisant le programme G qui sauvegarde les sommes gagnées par chaque joueur et par le casino. La somme positive gagnée par un client est ajoutée à sa carte.

Le Contrôleur met à jour deux tableaux : *client* et $jeux$.

Pour chaque client on sauvegarde les informations sur son identité et tous ses jeux.

La classe $jeux$ sauvegarde pour le jeu courant sa datation, les mises et les gains, le gain du casino inclus. L'historique des jeux est aussi sauvegardée.

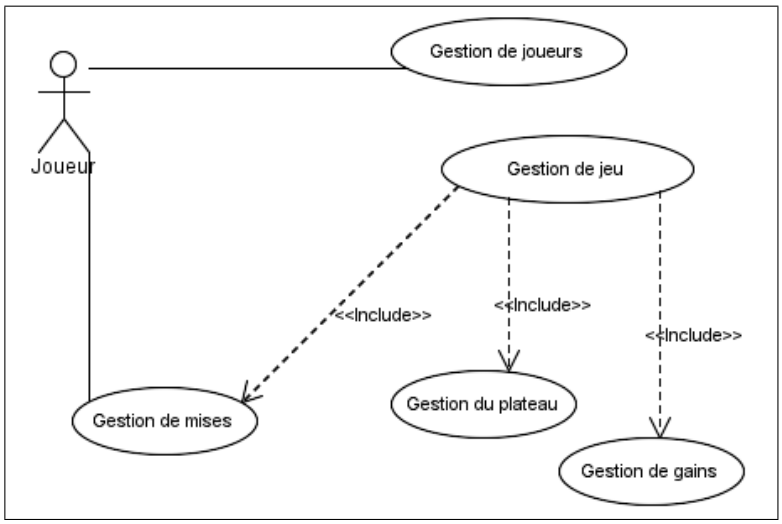
Réponse. Il n'y a pas de moyen pour dire que chaque client enregistré peut jouer (au bout de compte). Mais on peut exprimer, par exemple, les propriétés suivantes.

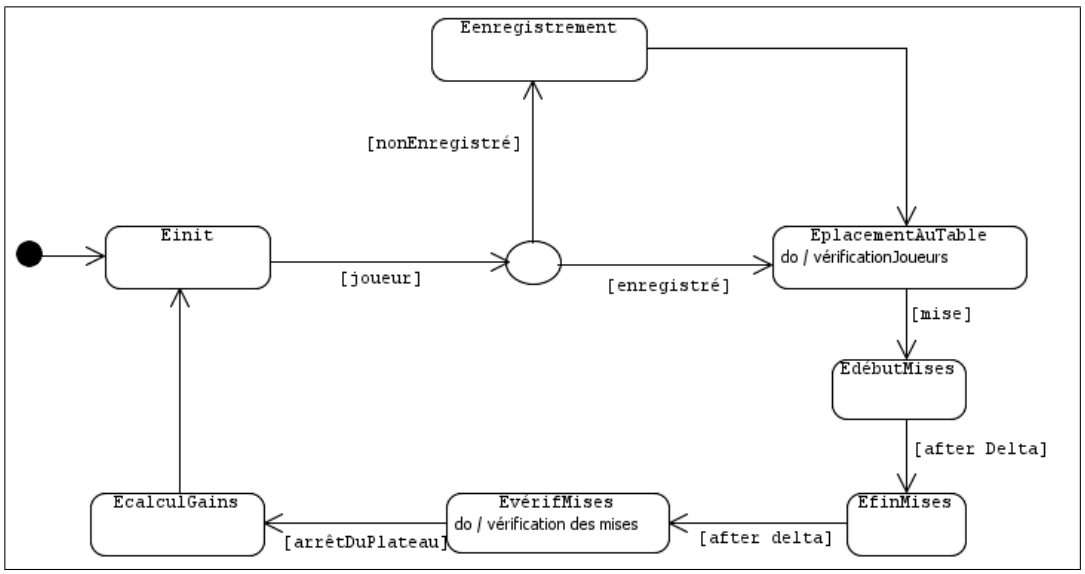
i. Chaque joueur (qui possède une place à la table) peut faire des mises.

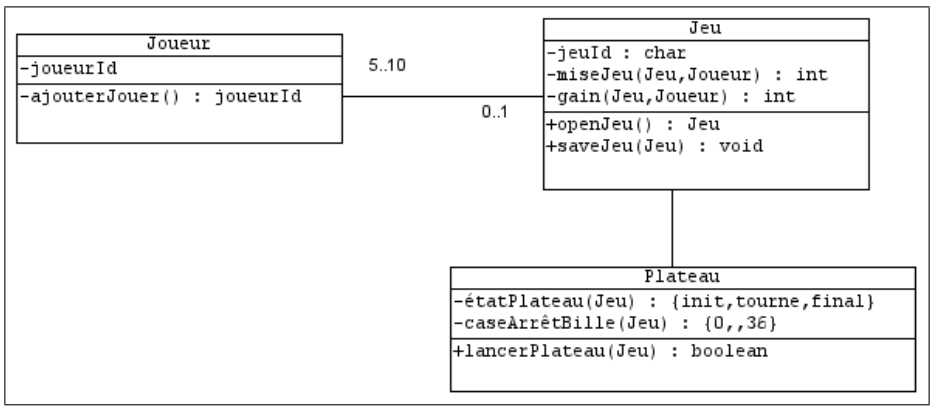
ii. Chaque joueur dont les mises sont positives et ne dépassent pas la somme sur sa carte a son gain calculé à la fin de jeu.

iii. Un joueur dont les mises sont zéro ou dépassent la somme sur sa carte est exclu de jeu.

Le mot "peut" dans i. est un connecteur modal. Une autre formulation de la même propriété sans modalité : si un joueur est installé à la table son clavier de mises sera débloqué. Pour être plus précis on peut ajouter le temps dans cette formulation.


Gestion de joueurs
Joueur
Gestion de jeu
<<Include>>
<<Include>>
<<Include>>
Gestion de mises
Gestion du plateau
Gestion de gains


Eenregistrement
[nonEnregistré]
Einit
[joueur]
[enregistré]
EplacementAuTable
do / vérificationJoueurs
[mise]
EdébutMises
[after Delta]
EcalculGains
EvérifMises
do / vérification des mises
EfinMises
[arrêtDuPlateau]
[after delta]


Joueur
-joueurId
-ajouterJouer() : joueurId
5..10
0..1
Jeu
-jeuId : char
-miseJeu(Jeu,Joueur) : int
-gain(Jeu,Joueur) : int
+openJeu() : Jeu
+saveJeu(Jeu) : void
Plateau
-étatPlateau(Jeu) : {init,tourne,final}
-caseArrêtBille(Jeu) : {0,,36}
+lancerPlateau(Jeu) : boolean

3 [5 points].
Écrivez une Abstract State Machine (ASM) de *base* pour le Contrôleur. Explicitez le vocabulaire de votre spécification.

N'oubliez pas qu'une ASM de base exécute ses opérateurs **if-then** de façon parallèle. Donc, une utilisation des drapeaux (qui en fait représentent les états de votre Contrôleur) peut faire les opérateurs séquentiels.

Pour mémoriser un moment de temps et assurer un délai Δ vous pouvez introduire un identificateur, disons, D avec la valeur initiale 0 et utiliser comme entrée l'identificateur du temps courant CT. Par exemple,

if $garde_0$ **then** $\ldots D := CT$
if $garde_1 \ \wedge \ CT = D + \Delta$ **then** $\ldots$

Vous pouvez utiliser J comme paramètre dans **forall** $J \in \{1, \ldots, r\}$ **do**. Pour calculer les gains après *stop* on utilise la fonction G sans aller dans les détails.

Réponse. À cette étape de spécification on voit que le texte donné est imprécise et incomplète. L'ASM ci-dessous précise la définition du problème. (On suppose que l'auteur de cette ASM a posé ses questions à la source de la définition du problème pour obtenir les précisions.)

Vocabulaire.

Sortes : $Places$, $\acute{E}tats = \{init, sMises, fMises, vMises, plateau, \}$, $Temps$ (les valeurs du temps ; le temps peut être interprété comme $\mathbb{Z}_{\geq 0}$ ou $\mathbb{R}_{\geq 0}$)

Fonctions :

$startMises :\rightarrow Bool$ (sortie)
$finMises :\rightarrow Bool$ (sortie)
$lancerPlateau :\rightarrow Bool$ (sortie)
$bloquerClavier :\rightarrow Bool$ (sortie)
$stop :\rightarrow Bool$ (entrée)
$somme : Places \rightarrow \mathbb{Z}$ (sortie, partagée avec l'environnement)
$G : Places \rightarrow void$ (une règle)
$CT :\rightarrow Temps$ (le temps courant (pensez à l'horloge d'ordinateur), entrée)
$state \rightarrow \acute{E}tats$ (états du contrôleur, interne)
$dl :\rightarrow Temps$ (moment de changement d'état, interne)
$\nu : Places \rightarrow (Places \cup \{undef\})$ (fonction interne pour exclure un joueur si ses mises ne sont pas bonnes)

Initialisation :
$startMise = finMises = lancerPlateau = bloquerClavier = false$,
$state = init$, $stop = false$, $\nu(J) = undef$ pour tout $J \in Places$.
les valeurs des autres fonctions ne sont pas importantes.

Programme d'ASM.

forall $J \in Places$

1 : **if** $state = init$ **then** $[\, startMises := true,\ state = sMises,\ dl := CT + \Delta \,]$

2 : **if** $state = sMises \wedge CT \geq dl$
then $[\, startMises := false,\ finMises := true,\ state := fMises,\ dl := CT + \delta \,]$

3 : **if** $state = fMises \wedge CT \geq dl$ **then** $[\, state := vMises,\ bloquerClavier := true \,]$

4 : **if** $state = vMises \wedge somme(J) \neq undef \wedge somme(J) \geq \sum_i mise(J,i) > 0$
then $[\, state := plateau,\ lancerPlateau := true,$
$somme(J) := somme(J) - \sum_i mise(J,i) \,]$

5 : **if** $state = vMises \wedge somme(J) \neq undef \wedge 0 < somme(J) < \sum_i mise(J,i)$
then $[\, somme(J) := 0,\ \nu(J) := J \,]$

6 : **if** $state = vMises \wedge somme(J) \neq undef \wedge \sum_i mise(J,i) = 0$
then $[\, somme(J) := somme(J) - \alpha,\ \nu(J) := J \,]$

7 : **if** $\nu(J) = J$ **then** $[\, somme(J) := undef,\ \nu(J) = undef]$

8 : **if** $state = plateau \wedge stop \wedge somme(J) \neq undef$
then $[\, G(J),\ state := init,\ lancerPlateau = finMises = bloquerClavier := false \,]$

On voit que le rôle de la fonction *somme* est mal définie : $somme(J)$ représente la somme sur la carte du joueur qui occupe la place J et ainsi que l'état de cette place (occupée ou non occupée). Mais le texte initial attribue ce rôle à *somme*.