Projet

But

Le but de ce projet consiste à implémenter le célèbre jeu du ***Uno.*** L'objectif principal est d'implémenter "la mécanique de base" qui correspond en définitive aux règles du jeu, aux actions des joueurs, à la gestion de leur score. L'interface graphique n'est pas l'objectif du projet !

Il vous faudra concevoir l'application pour qu'elle soit extensible. Et ce pour 2 raisons. La première raison est que, bien que le jeu du Uno standard soit de loin le plus connu, il existe des variantes avec des cartes ayant d'autres effets (carte permettant d'échanger sa main avec celle d'un autre joueur par exemple). Ces nouvelles cartes ne remettent pas en question la mécanique de base du jeu, donc il ne faudrait pas qu'ajouter un nouveau type de carte conduise à la réécriture de l'application ! La deuxième raison est que l'algorithme qui contrôle que la carte posée est légale, et qui applique éventuellement des effets à l'un ou l'autre des joueurs, n'est pas aussi simple qu'il n'y parait. De nombreux tests sont nécessaires et de nombreux paramètres (sens de rotation, qui est le prochain joueur, combien de cartes le joueur doit piocher, etc.) doivent simultanément être gérés. Réaliser cet algorithme du premier coup est un véritable défi ! Aussi, l'héritage (bien utilisé) vous permettra de progresser dans le développement sans modifier (ou en tout cas très peu) le code déjà développé et testé (une autre application de l'extensibilité). Le design pattern « Chaine de responsabilité » que nous étudierons en TD et en TP devra obligatoirement être mis en œuvre pour réaliser cet aspect du projet.

L'objectif final est donc de réaliser entièrement le jeu de Uno. Tout ce que vous devez développer se trouve dans les règles du jeu. Je vous demande de suivre la même règle qui celle que je vous donne et qui est sur Arche.

Pour conduire votre analyse du projet, et le développement que vous devez faire, je vous donne une liste de tests, rédigés en français. Vous devrez traduire ces tests sous forme d'instructions, lesquelles vont évidemment dépendre des choix de conception que vous allez faire. Ces tests décrivent précisément les règles que vous devez impérativement respectées. Et tout ce qui n'est pas dans les tests, n'a pas besoin d'être fait ou peut être fait à votre manière.

Déroulement du projet

Le projet fera l'objet d'un suivi lors des 3 dernières séances de TP (fin avril). Lors de ce suivi, il faudra proposer un diagramme de classes rapidement sur lequel nous pourrons déjà discuter et éventuellement corriger certaines maladresses de conception. Une soutenance intermédiaire aura lieu mi-mai et permettra de voir au moins en partie la réalisation des tests. Ce sera là aussi l'occasion de discuter voir de corriger certaines maladresses. La soutenance finale aura lieu le plus tard possible mais avant la date butoir pour rendre les notes de L2. Mais si vous avez terminé le projet avant cette date, il sera toujours possible d'anticiper la soutenance.

Soutenance intermédiaire

Pour la soutenance intermédiaire, aucune interface n'est utile. Tout se fera dans la console. Vous réaliserez un ou plusieurs programmes principaux destinés à réaliser une série de tests (voir plus loin). Les tests vous sont donnés. Vous devrez coder en dur les résultats attendus. Si l'exécution d'un test produit exactement le résultat attendu, alors le test est « passé ». Sinon, il a « échoué ». Lorsque tous les tests passent, le mécanique du jeu est construite.

Ces tests ne sont pas exhaustifs. Les cartes « Changement de sens » et « +4 » ne sont pas testées. Vous devez rédiger vous-même les tests garantissant que ces deux cartes sont bien prises en compte. Ce point fera l'objet d'une partie de l'évaluation.

Aucun rapport n'est exigé. Mais vous devrez montrer le diagramme de classes du projet. Un diagramme de classes doit tenir sur une feuille A4 et être lisible (pas plus de 12 classes sur une feuille A4). Si vous avez plus que 12 classes à montrer, alors découpez votre diagramme en plusieurs « sous diagrammes ». Le ou les diagrammes doivent être imprimés pour que nous puissions échanger, dessiner par-dessus, etc.
Ce diagramme de classes pourra être modifié au gré de vos développements.

Soutenance finale

La soutenance finale consistera à utiliser votre jeu dans sa version stable la plus aboutie (avec IHM si elle existe). Il s'agira de tester les différentes spécifications décrites précédemment. Aucun rapport n'est demandé mais vous devez montrer le diagramme de classes final de votre projet (qui aura sans doute évolué depuis la soutenance intermédiaire).

Démo

La démo n'aura d'intérêt qu'après la soutenance intermédiaire, qui elle ne contient pas d'interface graphique. Inutile de perdre du temps à modifier cette démo tant que votre mécanique de jeu n'est pas opérationnelle.

Sur Arche, vous trouverez un projet Eclipse contenant une démo. Vous pouvez naturellement partir de ce projet pour le compléter avec vos classes métier, votre interface personnalisée, etc. Mais vous ferez cela qu'après la soutenance intermédiaire.

Evaluation

Les critères pris en compte pour l'évaluation sont :

La validité : votre application doit faire ce qui est demandé ci-dessus. En particulier :

- Présentation du ou des diagrammes de classes
- Exécutions de toute la suite de tests
- Ajout des tests pour « Changement de sens » et « +4 »

La robustesse : votre application doit fonctionner correctement (pas de plantage, pas d'incohérence lors de l'exécution, etc.). Attention : ce qui compte avant tout, c'est un programme qui s'exécute ! Si vous n'avez finalement pas le temps d'implémenter toutes les règles du jeu, tant pis. Mais faites tourner quelque chose !

L'extensibilité : le modèle doit être bien conçu pour permettre un ajout facile de nouvelles cartes (pour étendre le jeu à des variantes), de nouvelles situations (pour implémenter progressivement les règles du jeu), de nouvelles interactions entre les décors, nouveaux niveaux. Cette qualité est évaluée en analysant l'organisation de vos classes. Elle ne se voit pas nécessairement à l'exécution. C'est ici que sera évalué la mise en œuvre de la chaine de responsabilité

La lisibilité : le code doit être organisé en packages appropriés, les classes doivent être bien structurées avec les constructeurs qui conviennent. Les getters et les setters, la fonction equals, toString etc. doivent être présents sauf s'il ne les faut pas. Le code doit être bien commenté, les identifiants doivent être bien choisis, etc. Des exceptions sont indispensables et être bien gérées. 5 classes centrales au projet doivent être entièrement documentées avec des commentaires javadoc et la documentation devra être générées.

Remarque générale : le projet se fait par deux. Si vous êtes seuls, parlez-en à votre enseignant qui vous attribuera un groupe.

Montrez uniquement un programme qui s'exécute. Un projet qui ne s'exécute pas, quoi qu'il contienne, recevra 0. Autant faire peu, bien et exécutable (vous aurez toujours plus que 0), que beaucoup, mal et pas exécutable.

L'objectif du projet est d'apprendre la programmation objet de manière pratique, et à passer l'examen final en étant plus à l'aise. Copier tout ou partie d'un autre projet, ou de copier un code trouvé sur Internet, ne vous apprendra rien et vous expose à de terribles sanctions. C'est un risque inutile. Laissez faire le projet par son binôme parce qu'il est plus à l'aise en programmation est un mauvais calcul. Lui va encore augmenter sa performance dans ce domaine, et vous, vous continuerez à ramer et avoir de mauvaises notes aux autres épreuves. C'est un très mauvais calcul…

Les tests

L'ensemble des tests ci-dessous, bien que n'utilisant que des cartes simples (numéro et couleur), permettent de valider l'essentiel de la « mécanique » du jeu. Par la suite, par extensions successives, nous ajouterons les autres types de cartes, avec les règles de vérification et les effets que ces cartes produisent. Pour autant, la « mécanique » du jeu ne devra pas être impactée par ces extensions. C'est tout le défi de bien organiser ses classes, de bien identifier où et comment mettre en place de l'héritage, bien identifier les exceptions, etc.

La réalisation de ces tests doit logiquement faire l'objet d'une fonction par test. Il sera sans doute important de bien « outiller » les classes pour pouvoir réaliser les tests facilement. Autrement dit, il sera utile d'ajouter aux classes des méthodes sans intérêt pour le jeu lui-même, mais très utiles (voire indispensables) pour l'écriture de ces tests.
Si un test passe alors il n'affiche rien, c'est-à-dire ni trace de debbugage, ni message, rien. Tout au plus on peut afficher à la fin du test « [OK] Test bla bla » ou « [NOT OK] Test bla bla », à l'instar de ce que vous voyez au démarrage de Linux. Si un test ne passe pas, c'est qu'une anomalie s'est produite et là, bien sûr, il faut afficher un message d'erreur détaillé « [ERREUR] alice bla bla bla ».
La suite des fonctions de test se conclue par un programme principal qui lance toute la série de tests. En théorie, une fois que tous les tests sont en place, la moindre modification de quelque chose dans l'application nécessite de rejouer l'ensemble de tests pour éviter que, par effet de bord, une modification qui paraît correcte ici n'engendre un gros plantage ailleurs.

Chaque test est décrit dans un environnement précis, c'est-à-dire avec une pile de cartes dans un ordre fixe et donné au début du test. Les fichiers mentionnés sont téléchargeables sur Arche dans le dossier du projet. Vous devez donc, au début de chaque test, faire en sorte que les joueurs possèdent bien les cartes prévues, que la pioche contienne bien dans l'ordre les cartes citées, et que la carte se trouvant au sommet du tas est bien celle indiquée. En principe, la liste des cartes dans le fichier est dans l'ordre correct pour qu'en distribuant les cartes une par une et en retournant la carte suivante, la situation soit celle décrite dans le jeu d'essai.

Tests coups légaux avec des cartes simples

Fichier de test : JeuTestCarteSimple.csv
La partie contient 3 joueurs : Alice, Bob et Charles
Distribution de 3 cartes, et Alice commence

Alice	*Bob*	*Charles*
--------	*--------*	*--------*
Vert 2	*Bleu 2*	*Bleu 9*
Jaune 6	*Jaune 4*	*Bleu 7*

Rouge 1 Rouge 9 Bleu 0

Tas : CarteSimple [valeur=8, couleur=Vert]

Pioche

CarteSimple [valeur=6, couleur=Jaune]
CarteSimple [valeur=4, couleur=Rouge]
CarteSimple [valeur=2, couleur=Vert]
CarteSimple [valeur=5, couleur=Bleu]
CarteSimple [valeur=0, couleur=Vert]

Alice joue une carte de la bonne couleur

Vérifier que le joueur courant est bien Alice
Vérifier que Alice possède bien 3 cartes
Alice joue le « 2 Vert »
Vérifier que Alice possède bien 2 cartes
Vérifier que les cartes d'Alice sont le « 6 jaune » et le « 1 rouge »
Vérifier que la carte au sommet du tas est le « 2 Vert »
Vérifier que le nombre de cartes du tas est 2
Alice finit le tour
Vérifier que le joueur courant est Bob

Bob joue une carte de couleur différente mais de même valeur

Vérifier que Bob possède bien 3 cartes
Bob pose le « 2 bleu »
Vérifier que Bob possède bien 2 cartes
Vérifier que les cartes de Bob sont le « 4 jaune » et le « 9 rouge »
Vérifier que la carte au sommet du tas est le « 2 Bleu »
Vérifier que le nombre de cartes du tas est 3
Bob finit le tour
Vérifier que le joueur courant est Charles

Tests coups illégaux avec des cartes simples

Tous ces tests vont lancer des exceptions. Toutes ces anomalies sont (en principe !) sanctionnées par 2 cartes de punitions, et fin du tour automatique si c'est le joueur courant qui commet la faute. C'est dans le catch de l'exception que doivent être réalisées ou non ces deux actions. Mais pour l'instant, il n'est pas utile de gérer la « punition ». Nous le ferons dans d'autres tests par la suite (une chose après l'autre).
Pour chacun des tests de cette partie, il faut réinitialiser la partie pour se retrouver dans les conditions des tests précédents

Test d'une carte illégale

Alice pose le « 6 jaune »

Vérifier dans le catch approprié que Alice possède toujours 3 cartes dont le « 6 Jaune »

Test d'un joueur qui pose deux cartes légales de suite

Alice pose le « 2 Vert » et finit son tour
Bob pose le « 2 Bleu » et finit son tour

Charles pose le « 6 Bleu » (RAS, c'est correct mais Charles ne finit pas le tour)
Vérifier que Charles possède 2 cartes
Charles pose le « 7 Bleu » (Carte légale mais il a déjà posé…)
Vérifier dans le catch approprié que Charles possède toujours 2 cartes dont le « 2 Bleu »

Test d'un joueur qui finit son tour sans rien faire

Alice finit son tour
Vérifier dans le catch approprié que Alice possède toujours 3 cartes

Test d'un joueur qui joue puis pioche

Alice joue le « 2 Vert » (RAS, le coup est légal)
Alice pioche
Vérifier dans le catch approprié que Alice possède toujours 2 cartes
Vérifier que la carte de la pioche est toujours le « 6 jaune »

Tests de la punition

Pour chacun des tests de cette partie, il faut réinitialiser la partie pour se retrouver dans les conditions des tests précédents

Test de la punition pour un coup illégal d'Alice (joueur courant)

Vérifier que le joueur courant est bien Alice
Alice pose le « 6 jaune » (coup illégal)
Dans le catch approprié :

- Punir Alice
- Vérifier que Bob est le joueur courant
- Vérifier que Alice possède 5 cartes dont le « 6 jaune » et le « 4 rouge » (les 2 cartes de la pioche)
- Vérifier que la prochaine carte de la pioche est le « 2 vert »

Test d'une action de bob lorsque ce n'est pas son tour

Vérifier que le joueur courant est bien Alice
Bob pioche (ce n'est pas son tour)
Dans le catch approprié :

- Punir Bob
- Vérifier que Alice est toujours le joueur courant

Vérifier que Bob possède 5 cartes dont le « 6 jaune » et le « 4 rouge » (les 2 cartes de la pioche)
Vérifier que la prochaine carte de la pioche est le « 2 vert »

Test du « Uno ! »

Une règle de base du jeu consiste à dire « Uno ! » dès qu'il nous reste plus qu'une seule carte en main. Bien sûr, il faut le dire quand c'est encore son tour…

Fichier de test : JeuTestCarteSimplePourUno.csv
Distribution de 2 cartes, et Alice commence

Alice	*Bob*	*Charles*
Vert 2	*Bleu 2*	*Bleu 9*
Jaune 6	*Jaune 4*	*Bleu 7*

Tas : CarteSimple [valeur=8, couleur=Vert]

Pioche
CarteSimple [valeur=6, couleur=Jaune]
CarteSimple [valeur=2, couleur=Vert]
CarteSimple [valeur=5, couleur=Bleu]
CarteSimple [valeur=0, couleur=Vert]
CarteSimple [valeur=3, couleur=Bleu]

Test lorsqu'Alice dit « Uno ! » au bon moment

Vérifier qu'Alice a bien 2 cartes
Alice pose le « 2 Vert »
Alice dit « Uno ! »
Alice finit son tour
Vérifier qu'Alice n'a plus qu'une seule carte
Vérifier que la carte au sommet du tas est le « 2 Vert »
Vérifier que le joueur courant est Bob

Test lorsqu'Alice oubli de dire « Uno ! »

Alice pose le « 2 Vert »
Alice finit son tour
Dans le catch approprié :
    Punir Alice
    Vérifier qu'Alice a maintenant 4 cartes
    Vérifier que la carte au sommet du tas est le « 8 Vert »
    Vérifier que le joueur courant est Bob

Test lorsque Bob dit « Uno ! » quand ce n'est pas son tour

Vérifier que Alice est le joueur courant
Bob dit « Uno ! »
Dans le catch approprié :
    Punir Bob
    Vérifier que Bob a maintenant 4 cartes
    Vérifier qu'Alice est toujours le joueur courant
    Vérifier que la carte au sommet du tas est le « 8 Vert »

Tous les tests ci-dessous permettent de vérifier que la mécanique du jeu est en place. Il ne manque plus qu'à ajouter les différents types de carte (passe ton tour, +2, +4, changement de couleur, et d'autres variantes encore). En principe, la chaine de responsabilités va permettre ces évolutions très facilement. Il ne sera plus nécessaire de retester tous les cas de figure ci-dessus.

Tests avec des cartes « Passe ton tour »

Fichier de test : JeuTestCartePasser.csv
La partie contient 3 joueurs : Alice, Bob et Charles
Distribution de 3 cartes, et Alice commence

Alice	*Bob*	*Charles*
Rouge Passe	*Jaune 6*	*Bleu 1*
Bleu 9	*Vert 6*	*Vert Passe*
Jaune 4	*Bleu 7*	*Rouge 1*

Tas : CarteSimple [valeur=9, couleur=Rouge]

Pioche

CarteSimple [valeur=0, couleur=Bleu]
CarteSimple [valeur=8, couleur=Vert]
CarteSimple [valeur=2, couleur=Vert]
CarteSimple [valeur=4, couleur=Rouge]
CarteSimple [valeur=2, couleur=Vert]

Test de coups légaux avec des cartes « Passe ton tour »

Vérifier que Alice est bien le joueur courant
Alice pose le « Passe ton tout rouge »
Alice finit son tour
Vérifier que Charles est le joueur courant
Vérifier que la carte du tas est bien le « Passe ton tour rouge »

Charles pose le « Passe ton tour vert »
Charles finit son tour
Vérifier que Bob est le joueur courant

Vérifier que la carte du tas est bien le « Passe ton tour vert »

Bob pose le « 6 Vert »
Bob finit son tour
Vérifier que Charles est le joueur courant
Vérifier que la carte du tas est bien le « 6 Vert »

Test d'une carte simple illégale sur un « Passe ton tour »

Vérifier que Alice est bien le joueur courant
Alice pose le « Passe ton tout rouge »
Alice finit son tour

Vérifier que Charles est le joueur courant
Vérifier que Charles possède bien 3 cartes
Charles pose le « 1 Bleu »
Charles finit son tour
Vérifier dans l'exception appropriée que Charles a toujours 3 cartes

Test d'un « Passe ton tour » illégal sur une carte simple

Vérifier que Alice est bien le joueur courant
Alice pose le « 9 bleu »
Alice finit son tour
Bob pose le « 7 Bleu »
Bob finit son tour

Vérifier que Charles est le joueur courant
Vérifier que Charles possède bien 3 cartes
Charles pose le « Passe ton tour vert »
Charles finit son tour

Vérifier dans l'exception appropriée que Charles a toujours 3 cartes

Tests avec des cartes « +2 »

Test d'un coup légal avec une carte « +2 »

Fichier de test : JeuTestCartePlus2.csv
Distribution de 3 cartes, et Alice commence

Alice	*Bob*	*Charles*
Vert Plus2	*Jaune 6*	*Bleu 1*
Bleu 9	*Vert 6*	*Vert Plus2*

Jaune 4 Bleu 7 Vert 1

Tas : CarteSimple [valeur=9, couleur=Vert]

Pioche

CarteSimple [valeur=0, couleur=Bleu]
CarteSimple [valeur=8, couleur=Vert]
CarteSimple [valeur=2, couleur=Vert]
CarteSimple [valeur=4, couleur=Rouge]
CarteSimple [valeur=2, couleur=Vert]

Vérifier qu'Alice est le joueur courant
Alice pose « +2 Vert »
Alice finit son tour
Vérifier que Bob est le joueur courant
Vérifier que Bob possède 3 cartes
Bob doit encaisser l'attaque (donc piocher 2 cartes et finir automatiquement son tour)
Vérifier que Bob possède 5 cartes
Vérifier que Charles est le joueur courant
Charles pose le « 1 Vert »
Charles finit son tour
Vérifier que Charles possède 2 cartes

Test d'un coup légal avec cumul de cartes « +2 »

Vérifier qu'Alice est le joueur courant
Alice pioche une carte
Alice finit son tour
Vérifier que Bob est le joueur courant
Bob pioche une carte
Bob finit son tour
Vérifier que Charles est le joueur courant
Charles pose le « +2 Vert »
Charles finit son tour
Vérifier que Alice est le joueur courant
Alice pose le « +2 Vert »
Alice finit son tour
Vérifier que Bob est le joueur courant
Vérifier que Bob possède 4 cartes
Bob encaisse l'attaque (donc pioche 4 cartes et finit son tour automatiquement)
Vérifier que Bob possède 8 cartes
Vérifier que Charles est le joueur courant