Projet

But

Le but de ce projet consiste à implémenter le jeu ***Boulder Dash***. Évidemment, il ne sera pas question d'implémenter toutes les possibilités qu'offre le jeu original. L'objectif est d'implémenter "la mécanique de base", avec une conception suffisamment extensible pour rajouter de nouvelles fonctionnalités (nouveaux décors, nouveaux comportements, etc.). Le projet comporte une partie "métier" (les fonctionnalités de base du jeu) et une partie IHM (néanmoins assez limitée) qui ne pourra pas être traitée tout de suite. C'est pourquoi, il est prévu une soutenance intermédiaire pour évaluer votre travail sur la partie métier.
Une démo vous sera fournie. Il s'agit d'une interface graphique opérationnelle qui permet de déplacer Rockford dans un plateau de démonstration. Vous aurez donc une base pour compléter l'interface et rendre le jeu totalement opérationnel. Elle contient également d'autres fichiers utiles pour développer la partie graphique du projet (mais ce n'est pas la priorité !).

Figure 1 : exemple de plateau

Le principe du jeu

Rockford est une créature qui vit sous terre. Il creuse des galeries dans lesquelles il peut se déplacer. Il creuse ces galeries pour trouver des diamants ! Mais parfois, il creuse sous un rocher qui peut lui tomber dessus et l'écraser. Parfois, il rencontre d'autres créatures qui se déplacent également dans les galeries. Si une créature attrape Rockford, il meurt… S'il est suffisamment malin, Rockford peut faire exprès de faire tomber un rocher sur une créature. Elle se transforme alors en…diamant ! Il existe de nombreuses interactions dont les résultats sont variés. C'est pourquoi, toute la subtilité du projet consiste à faire en sorte qu'il soit facilement extensible. L'ajout d'une nouvelle situation (une nouvelle interaction entre deux objets) ne doit pas nécessiter la modification de ce qui existait avant. Le programme doit rester opérationnel. Il ne doit pas contenir de switch ou de longues conditionnelles pour traiter les différentes situations…

Spécifications détaillées

Rockford : gentille bestiole qui se déplace horizontalement ou verticalement. Ce qui se passe lors du déplacement de Rockford dépend de ce que contient la case vers laquelle il se déplace :

- Vide : il ne se passe rien. Il change de case. La case où il se trouvait devient vide, celle qui était vide contient Rockford
- Terre : la terre disparait. A la place il y a Rockford. Et la case où se trouvait Rockford devient vide. Lorsque Rockford quittera la case, elle sera vide aussi (plus de terre)
- Diamant : C'est comme si la case contenait de la terre mais le compteur de diamant de Rockford augmente de 1
- Monstre : Le monstre disparait, mais Rockford perd une vie. Si c'était sa dernière vie, c'est « Game Over »
- Rocher : si la case de l'autre côté du rocher est vide, alors Rockford déplace le rocher. Si elle n'est pas vide, alors il ne se passe rien. Le déplacement n'a pas lieu
- Acier, bord du plateau ou toute autre chose : il ne se passe rien

Ce qui se passe lorsqu'un Rocher tombe par gravité dépend aussi du contenu de la case où le rocher tombe :

- Vide : le rocher tombe d'une case. La case où se trouvait le rocher devient vide
- Rockford : si le rocher était immobile (Rockford vient de creuser en dessous à l'instant) alors le rocher ne bouge pas. Mais si le rocher était en mouvement, alors Rockford est écrasé et meurt. Donc il perd une vie. Mais s'il reste encore au moins une vie à Rockford, alors le rocher disparaît et Rockford continue la partie. Sinon, « Game over »
- Monstre : Le rocher ne se déplace pas, mais à la place du monstre apparaît un diamant
- Rocher : lorsqu'un rocher tombe sur un autre rocher, il peut rouler sur le côté. Selon ce qui s'y trouve, le rocher continue sa course....
- Acier ou bord du plateau ou toute autre chose : il ne se passe rien

Ce qui se passe lorsqu'un Diamant tombe par gravité dépend aussi du contenu de la case où il tombe. Un diamant ne peut pas rouler :

- Vide : le diamant tombe d'une case. La case où se trouvait le diamant devient vide
- Rockford : si le diamant était immobile (Rockford vient de creuser en dessous à l'instant) alors le diamant ne bouge pas. Mais si le diamant était en mouvement, alors Rockford est écrasé et meurt. Donc il perd une vie. Mais s'il reste encore au moins une vie à Rockford, alors le diamant disparaît et Rockford continue la partie avec un diamant de plus. Sinon, « Game over »
- Bord du plateau ou toute autre chose : il ne se passe rien

Vous ne pourrez pas implémenter toutes ces interactions du premier coup. Il faut prévoir une architecture où vous ajouterez les interactions les unes après les autres, sans pour autant modifier ce qui a déjà été développé. Le design pattern « Chain Of Responsabilities » (Chaine de responsabilités) permet d'atteindre cet objectif. Il sera étudié par ailleurs et vous devrait transposer obligatoirement ce pattern dans le contexte de ce projet.

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 sur la grille donnée dans la démo (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 ».

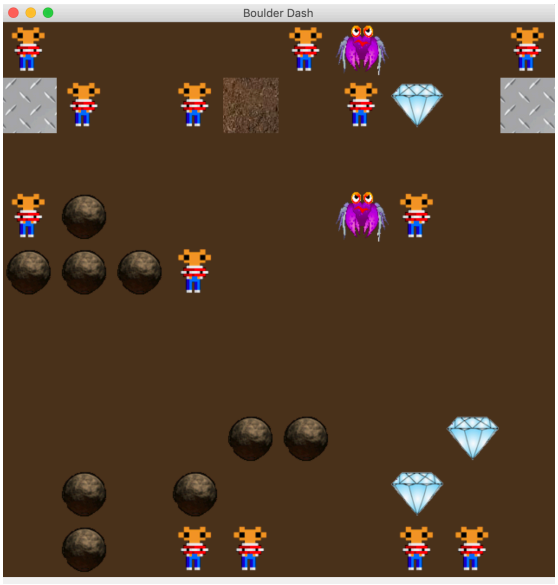

Figure 2 : plateau de test

Tester les déplacements :

Déplacer Rockford(1,1) vers la case contenant du vide (1,2)
Résultat attendu : la case (1,1) contient du vide, la case (1,2) contient Rockford
Déplacer Rockford(2,2) vers la case contenant de l'acier (2,1)
Résultat attendu : la case (2,2) contient Rockeford et la case (2,1) contient de l'acier
Déplacer Rockford(1,6) vers la case contenant un monstre (1,7)
Résultat attendu : la case (1,6) contient du vide, la case (1,7) contient Rockford, Rockford a une vie de moins
Déplacer Rockford(2,4) vers la case contenant de la terre (2,5)
Résultat attendu : la case (2,4) contient du vide, la case (2,5) contient Rockford
Déplacer Rockford(2,7) vers la case contenant un diamant (2,8)
Résultat attendu : la case (2,7) contient du vide, la case (2,8) contient Rockford, Rockford a un diamant de plus
Déplacer Rockford(4,1) vers la case contenant un rocher (4,2)
Résultat attendu : la case (4,1) contient du vide, la case (4,2) contient Rockford et la case (4,3) contient le rocher
Déplacer Rockford(5,4) vers la case contenant un rocher (5,3)
Résultat attendu : la case (5,4) contient Rockford, la case (5,3) contient un rocher

Les déplacements de Rockford ci-dessous sont détaillés. Pour les tests suivants, nous ne donnerons que l'intention. Vous devrez traduire cette intention par une série de déplacements et de contrôles dans l'esprits des tests sur Rockford

Faire tomber le rocher (8,6) jusqu'à ce qu'il s'arrête sur le bord du plateau
Faire tomber le rocher (8,5) jusqu'à ce qu'il rencontre Rockford qui doit perdre une vie
Faire tomber le rocher (9,4) vers le bas. Rien ne doit changer
Faire tomber le rocher (9,2) vers le bas. Il doit rouler vers la droite et se retrouver en (10,3)

Faire tomber le diamant (9,8) sur Rockford. Il ne doit rien se passer
Faire tomber le diamant (8,9) sur Rockford. Il doit disparaitre, en faisant gagner un diamant à Rockford et lui faire perdre une vie

Cette liste de tests ne décrit pas tout ce que le jeu original permet de faire. Néanmoins, si tous ces tests passent, vous aurez un jeu qui commencera à ressembler à quelque chose. La liste peut évidemment être complétée avec de nouvelles situations que vous voulez prendre en compte dans votre projet.

Vous devez commencer par coder ces tests, c'est-à-dire avant de coder les maillons de la chaine de responsabilités qui implémentera ces déplacements. Vous devez faire en sorte que ces tests soient compilables de manière à pouvoir les exécuter. Bien entendu, puisque la chaine de responsabilités est vide, tous les tests échoueront. Alors seulement, vous pourrez ajouter un maillon dans la chaine de responsabilité et vérifier que certains tests passeront, et pas d'autres. Puis, un autre maillon, et encore un autre, etc.

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 en particulier). 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 (Figure 2). En fait, il s'agit essentiellement du canvas dans lequel sont dessinées les images des bonbons constituant une grille. L'interface de cette démo est opérationnelle et c'est son principal intérêt. La mécanique du "clavier" est en place, ainsi que le timer affichant le temps qui s'écoule. Vous pouvez naturellement partir de ce projet pour le compléter avec vos classes métier, votre interface personnalisée, etc.

Evaluation

Les critères pris en compte pour l'évaluation sont :

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

La robustesse : votre application doit fonctionner correctement (pas de plantage, pas d'incohérence lors de l'exécution, etc.).

L'extensibilité : le modèle doit être bien conçu pour permettre un ajout facile, de nouveaux décors, 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.

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 (javadoc sera testé lors de la soutenance), les identifiants doivent être bien choisis, etc. Des exceptions sont indispensables et être bien gérées.

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.

Soutenance : la soutenance intermédiaire aura lieu à mi-parcours, c'est-à-dire la semaine du 27 avril (après les vacances de printemps). La soutenance finale aura lieu le plus tard possible, c'est-à-dire aux environs de la semaine d'examen.