INSA RENNES

Institut National des Sciences Appliquées
Ecole publique d'ingénieurs

Projet
Analyse, Conception et Programmation Orientée Objet
Quatrième année - Informatique

2010-2011

Éric Anquetil, Département Informatique
eric.anquetil@insa-rennes.fr
Peggy Cellier, Département Informatique
peggy.cellier@insa-rennes.fr
Gabriel Cirio, Département Informatique
gabriel.cirio@irisa.fr
Pascal Garcia, Département Informatique
pascal.garcia@insa-rennes.fr
Laurent George, Département Informatique
l.george@inria.fr
Maud Marchal, Département Informatique
maud.marchal@insa-rennes.fr
Léo Terziman, Département Informatique
leo.terziman@inria.fr

Table des matières

Consignes générales pour le projet

Vous avez 6 séances prévues pour le projet. La dernière séance est une séance de finalisation et de préparation de la soutenance.

En ce qui concerne le travail à rendre, il se fera en 2 phases.

1 Rapport de conception

Vous devrez rendre votre rapport de conception au plus tard le **30 novembre 2010 avant minuit** sur la plateforme moodle (aucun rapport envoyé par mail ne sera pris en compte pour la notation).

2 Soutenance de projet

Vous devrez aussi préparer une soutenance de 10 min. Les soutenances auront lieu le **vendredi 7 janvier 2011 matin**. Cette soutenance sera centrée sur une démonstration de votre jeu. Prévoyez plusieurs cas d'utilisation.

Il est aussi impératif qu'au plus tard le **jeudi 6 janvier 2011 à 17h**, vous ayez déposé sur la plateforme moodle les documents suivants :

- la documentation utilisateur de votre jeu,
- le code source (en entier, commenté et avec les tests),
- un exécutable qui fonctionne.

IMPORTANT : Chaque jour de retard entrainera un point de pénalité sur la note finale.

Règles du jeu "Canon Noir"

1 Composition du jeu

Le jeu Canon Noir est composé des éléments suivants :

- **un plateau de jeu** représentant une partie de la mer des Caraïbes ; La description des éléments obligatoires appartenant au plateau est la suivante :
  - le plateau est décomposé en cases : la longueur est constituée de 11 cases, la largeur de 8 cases ;
  - **2 îles** sont représentées ;
  - il existe **4 ports** de 4 couleurs différentes, situés aux 4 coins du plateau ;
  - **4 cases TRESOR** et **4 cases CANON NOIR** sont distribuées sur le plateau ;
- **2 dés** à 6 faces ;
- **12 pions** représentant les navires des joueurs :
  - 3 navires par joueur, les voiles indiquent la couleur du joueur ;
  - 3 types de navire : caravelle (3 mâts), frégate (2 mâts) et radeau.
- **un canon** et 4 boulets ;
- 14 trésors disposés sur les 4 cases TRESOR ;

2 Principe du jeu

Chaque joueur est le commandant d'une flotte. Il choisit la couleur de son port et de sa flotte.

Le gagnant du jeu est la première personne qui ramène 3 trésors à son port. L'objectif du jeu est donc d'atteindre une des cases TRESOR, prendre un trésor sur son navire et le ramener au port, 3 fois de suite.

Le jeu comporte néanmoins également des cases CANON NOIR. A partir de ces cases, un joueur peut tirer sur ses adversaires avec le canon afin de les empêcher de retourner à leur port avec un trésor. Pour tirer un boulet de canon, le joueur doit néanmoins éviter le relief des îles.

3 Début de la partie

3.1 Nombre de joueurs

Le jeu Canon Noir peut se jouer à 4 joueurs maximum : chaque joueur possède alors 1 flotte. A 3 joueurs, une flotte est laissée de côté. A 2 joueurs, chaque joueur prend 2 flottes.

3.2 Début du jeu

Au début du jeu, chaque joueur choisit la couleur de son port et de sa flotte. Le joueur qui commence est celui qui obtient le meilleur score après le lancer des 2 dés. Les autres joueurs partent dans l'ordre de leur points au lancer de dés. Au départ, chaque commandant met en jeu sa caravelle qui part du port correspondant à sa couleur.

4 Gestion de la flotte

4.1 Déplacement des bateaux

Les déplacements des bateaux se font à l'aide des dés. Les déplacements peuvent s'effectuer dans n'importe quelle direction (horizontalement, verticalement ou diagonalement) mais il est interdit de faire un aller-retour ou de bifurquer lors d'un déplacement. Pour résumer, un déplacement s'effectuera dans une seule direction en utilisant la totalité des points des dés.

Par conséquence, il faut tomber piler sur une case TRESOR ou CANON NOIR si on souhaite l'atteindre.

Les îles constituent des obstacles aux déplacements. Elles doivent être contournées mais il est possible d'utiliser les cases sur lesquelles elles empiètent.

4.2 Gestion des différents bateaux

- **Caravelle** :
  le joueur commence avec sa caravelle qui va partir du port correspondant à sa couleur. Pour faire avancer une caravelle, le joueur lance les 2 dés puis choisit d'utiliser le total des points des 2 dés ou bien les points d'un dé uniquement. Si la caravelle est coulée par un adversaire, le commandant la remplace par une frégate.
- **Frégate** :
  Lors qu'une caravelle est coulée, elle est remplacée par une frégate. Le trésor transporté par la caravelle est conservé mais la frégate ne peut se déplacer qu'avec un seul dé.
- **Radeau** :
  Si la frégate est coulée, le commandant doit rentrer au port avec un radeau et donner le trésor qu'il transporte (éventuellement) au joueur qui a coulé la frégate. Un seul dé est utilisable pour déplacer un radeau. Un radeau ne peut pas transporter de trésor.
  Quand le joueur a rejoint le port, il a le droit au tour suivant de repartir avec sa caravelle et d'utiliser de nouveau les 2 dés en même temps.

5 Gestion du canon

Il existe 2 possibilités pour couler un navire ennemi :

- **Première possibilité : arriver pile sur une case CANON NOIR**
  Le commandant prend alors le canon, le met sur la case correspondante, charge le canon, vise son adversaire et tire. S'il réussit son tir, cet adversaire change de bâtiment dans sa flotte. S'il le manque, le jeu continue.
- **Deuxième possibilité : utiliser les cases en bordure du jeu**
  Le commandant peut positionner son bateau en bordure du jeu, exactement en face et à l'opposé d'un adversaire. Dans ce cas, il engage un duel au canon (le joueur A est celui qui engage le duel, le joueur B est celui qui est attaqué) :

– A prend le canon, le charge et tire sur B ;
– S’il réussit son tir, B doit changer de navire et le duel s’arrête. B donne son trésor s’il a perdu une frégate.
– S’il le manque, il remet son navire dans sa position initiale et c’est B qui prend le canon et tire.
– S’il réussit son tir, A change de navire. S’il possède une frégate et transporte un trésor, il le donne à B.
– S’il le manque, le combat s’arrête là.

Modélisation (Séances 1 et 2)

1 Conception du jeu

Le TD du 15 novembre est consacré à la modélisation du projet (diagrammes de classes et Design Patterns que vous pouvez utiliser).

Les thèmes essentiels que vous devez aborder lors de la phase d'analyse et de conception sont les suivants :

- Modélisation du moteur du jeu ;
- Gestion des événements entre la partie graphique et le moteur du jeu ;
- Gestion de l'affichage et réflexion sur les Design Patterns associés ;
- Gestion de la carte du jeu (les différents types de cases et les informations sur les joueurs).

Afin de vous aider dans votre conception, vous pouvez envisager de développer les scénarios suivants :

- Lancer des dés, choix du nombre de dés et détermination des cases accessibles ;
- Gestion du canon ;
- Déroulement d'un duel.

Il existe également de nombreux autres scénarios que vous pouvez développer (scénarios nominaux ou bien gestion des erreurs notamment).

2 Modélisation UML

Votre modélisation du jeu à l'aide de diagrammes UML sera basée sur votre analyse. Votre rapport de conception comportera notamment les parties suivantes :

1. Illustration des fonctionnalités du jeu à l'aide de cas d'utilisation.
2. Diagrammes de classe représentant :
   - la modélisation globale du jeu ;
   - les différents modules du jeu : moteur, gestion de l'affichage, gestion de la carte, etc. avec la mise en valeur des Design Patterns associés.
3. Diagrammes d'activités et d'états-transitions pour modéliser le fonctionnement des différents modules.
4. Diagrammes d'interaction avec le détail des différents scénarios que vous aurez choisis. Il faut au moins modéliser un scénario nominal, un scénario alternatif et un scénario avec des erreurs.

5. Diagrammes de composants et de déploiement : ces diagrammes seront réalisés dans la dernière partie de la phase de modélisation avant de passer à la génération de code.

3 Rapport de conception

Votre rapport de conception est une synthèse de votre modélisation du jeu "Canon Noir". Il comportera les différentes parties suggérées dans la section précédente. Vous pouvez le compléter avec tout autre élément nécessaire à la compréhension de votre modélisation.

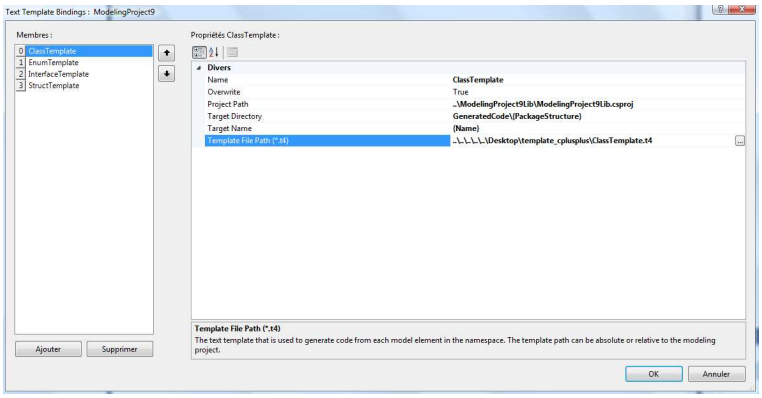
Génération du code (Séance 3)

La génération automatique de code permet de convertir un projet de modélisation en code source. Avec ses outils de modélisation, Visual Studio permet de synchroniser le code source et le modèle UML d'un projet. Il permet ainsi la génération automatique de code et la mise à jour du diagramme de classes suite à une modification du code source. Ceci permet au développeur d'avoir toujours accès à une vue du modèle de son projet.

Ces fonctionnalités sont disponibles pour les langages .NET C# et VB. Ceci n'est pas le cas pour le C++. Néanmoins, grâce au pack de modélisation et de visualisation, il est possible d'implémenter ses propres templates de génération de code source en utilisant la syntaxe T4 (Text Transformation Template Toolkit) de Microsoft. Nous vous fournissons ces templates, permettant de générer le code source d'un projet simple à partir de sa modélisation UML. Cependant, ces templates ne permettent pas la mise à jour automatique du projet de modélisation suite à des modifications du code source.

Voici les étapes à suivre pour utiliser ces templates :

1. Avant de lancer Visual Studio, vérifiez si le pack "Visualization and Modeling Feature Pack" est bien installé.
2. Dans le diagramme de classes de votre projet de modélisation, faites un clic droit sur la fenêtre et sélectionnez *Generate Code*. Lors du premier lancement, un message vous demandera si vous voulez configurer l'outil de génération de code. Cliquez sur *Oui*.
3. Pour chaque item (ClassTemplate, EnumTemplate, InterfaceTemplate, StructTemplate), choisissez le fichier correspondant dans l'entrée *Template File Path*. Ces chemins d'accès peuvent être modifiés ultérieurement en passant par le menu *Architecture->Configure Default Code Generation Settings*.



4. Le code source généré apparaît dans le dossier GeneratedCode sous le projet de modélisation.

La génération du code (squelettes des classes) à partir de votre projet de modélisation est à effectuer lors de la séance 3. Vous pourrez ensuite compléter lors de cette séance vos différentes classes à l'aide de votre modélisation dynamique réalisée dans les séances précédentes.

Aspect graphique du projet (Séance 4)

1 Description des vues

L'interface graphique se décompose principalement en deux vues distinctes ayant chacune leurs spécificité correspondant aux deux phases distinctes du jeu.

1.1 Vue du plateau

Il s'agit de la vue principale du jeu. Dans celle-ci, le plateau de jeu (figure 1) est utilisé pour représenter l'état actuel du jeu. Le plateau est lui même subdivisé en cases sur lesquelles les différents éléments du jeu (navires, trésors et canons) sont représentés.

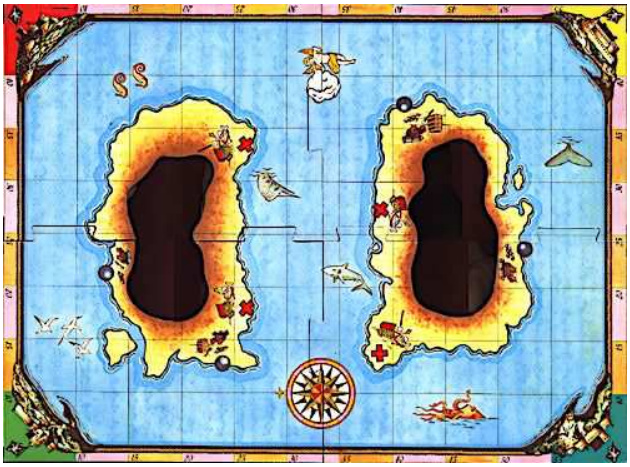


FIG. 1 – Plateau de jeu

Les cases doivent pouvoir être sélectionnées à l'aide de la souris afin de choisir la destination du navire du joueur à chaque tour. Les cases accessibles doivent être mises en évidence afin que le joueur puisse choisir aisément la case de destination.

De plus, cette vue doit également faire apparaître un moyen de tirer les dés (1 ou 2), ainsi que le résultat obtenu.

Lors de l'arrivée sur une case trésor, l'icône représentant le navire doit également être mise a jour afin d'indiquer les nouvelles propriétés de celui-ci.

De même, à l'arrivée sur une case canon, l'utilisateur doit avoir la possibilité de sélectionner la direction du tir soit en sélectionnant une case, soit en utilisant le clavier par exemple. Une fois la direction choisie, l'interface doit basculer dans la deuxième vue afin de permettre le tir.

1.2 Vue utilisée lors des tirs

Dans cette vue, le plateau de jeu est représenté de manière transversale suivant la direction du tir précédemment sélectionnée. La hauteur (altitude) de chacune des cases sur le trajet doit être représentée (coupe de terrain).

Plusieurs possibilités s'offrent à vous sur la manière de représenter cette coupe de terrain. La plus simple est de la représenter sous la forme d'un histogramme. Cependant, il va vous falloir calculer la liste des cases à traverser, ainsi que la largeur affectée à chaque case dans l'histogramme (la distance parcourue dans chaque case est variable). La figure 2 illustre ce principe.

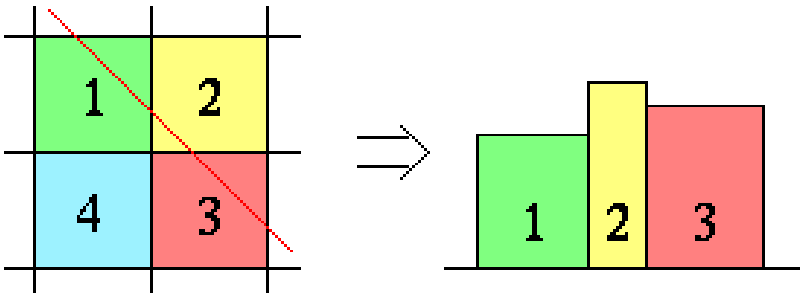


FIG. 2 – Calcul de la coupe transversale

Afin de résoudre ce problème, une solution consiste à calculer l'équation cartésienne de la direction du tir dans le plan du plateau, puis de calculer l'intersection de cette droite avec les droites formées par les limites entre les cases. Une fois les intersections connues, vous pouvez alors calculer la longueur (norme euclidienne) entre deux points d'intersection.

Soit xOy le repère orthonormé formé par le plateau de jeu, tel que chaque case soit de dimension 1×1, alignées sur le repère, en prenant le coin inférieur gauche du plateau comme origine du repère. Soit (i_1, j_1) les coordonnées de la case du canon et (i_2, j_2) celles du navire visé, avec $i \in [1, 11]$ et $j \in [1, 8]$. On a donc :

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} i_n - \frac{1}{2} \\ j_n - \frac{1}{2} \end{pmatrix}$$

Le plan de coupe est donc défini par la droite d'équation :

$$f(x) = \frac{(y_2 - y_1)x + (x_2 y_1 - x_1 y_2)}{x_2 - x_1}$$

Et son inverse est donné par :

$$g(y) = \frac{(x_2 - x_1)y - (x_2 y_1 - x_1 y_2)}{y_2 - y_1}$$

La liste des cases traversées peut donc être calculée de la manière suivante :

```
// coordonnees des intersections avec les limites entre cases
std::list<std::pair<double, double> > intersections;
// donnees de l'histogramme
std::list<std::pair<double, double> > histogramme;
std::list<std::pair<double, double> >::iterator it, end;
double h, l, xa, ya, xb, yb;
unsigned int x, y, i, j;
int inc_x, inc_y;

if(x1 < x2) {
  inc_x = 1;
  x = ((unsigned int) x1) + 1;
} else {
  inc_x = -1;
  x = (unsigned int) x1;
}

if(y1 < y2) {
  inc_y = 1;
  y = ((unsigned int) y1) + 1;
} else {
  inc_y = -1;
  y = (unsigned int) y1;
}

while(x * inc_x < x2 * inc_x && y * inc_y < y2 * inc_y) {
  if (f(x) == y) {
    intersections.push_back(std::pair<double, double>(x, y));
    x += inc_x;
    y += inc_y;
  } else if(f(x) * inc_y < y * inc_y) {
    intersections.push_back(std::pair<double, double>(x, f(x)));
    x += inc_x;
  } else {
    intersections.push_back(std::pair<double, double>(g(y), y));
    y += inc_y;
  }
}

if(intersections.size() >= 1) {
  xb = x1;
  yb = y1;

  intersections.push_back(x2, y2);

  it = intersections.begin();
  end = intersection.end();

  for(; it != end; ++it) {
    xa = xb;
    ya = yb;
    xb = it->first;
    yb = it->second;

    i = (unsigned int) min(xa, xb) + 1;
    j = (unsigned int) min(ya, yb) + 1;

    h = hauteur(i, j);
    l = sqrt((xb - xa) * (xb - xa) + (yb - ya) * (yb - ya));
```

```
    histogramme.push_back(std::pair<double, double>(l, h));
  }
}

return histogramme;
```

Cette vue devra également faire apparaître les différents éléments présents sur les cases traversées, en particulier les navires. Le joueur devra alors sélectionner une direction pour son tir (angle), ainsi que la puissance du tir en pressant plus ou moins longuement sur une touche par exemple. La trajectoire du boulet de canon devra alors être calculée suivant les équations d'une trajectoire parabolique. En prenant le canon comme origine, pour une vitesse initiale telle que :

$$V = \begin{pmatrix} V_x \\ 0 \\ V_z \end{pmatrix}$$

On obtient l'équation suivante :

$$z(x) = -\frac{1}{2}\frac{g}{V_x^2}x^2 + \frac{V_z}{V_x}x$$

avec $g = 9.81 m.s^{-2}$ l'accélération de la pesanteur.

Notez que pour conserver l'intérêt du jeu, l'utilisateur ne devra voir la trajectoire produite qu'une fois le tir effectué. De plus, vous devrez penser à gérer les éventuelles collisions entre le boulet de canon et le relief du plateau de jeu. Pour cela, il vous suffit de parcourir l'histogramme :

```
std::list<std::pair<double, double> >::iterator it, end;
bool colision = false;
double x, h;

if(histogramme.size() > 1) {
  it = histogramme.begin();
  end = histogramme.end();

  h = it->second;
  x = it->first
  ++it

  for(; it != end && ! colision; ++it) {
    if(z(x) <= h || z(x) <= it->second) {
      colision = true;
    }
    x += it->first;
  }
}

return colision;
```

2 Ressources disponibles

L'ensemble des exemples vus en cours sur la partie graphique du projet est disponible sur le Moodle.

Tests (Séance 5)

Le petit résumé qui est fait sur les tests ne remplace pas un cours sur le test mais vous donne/rappelle quelques notions sur le test.

1 Rappel sur les tests

Dans le cadre d'une bonne gestion de projet, un projet informatique s'accompagne de plusieurs documents comme les spécifications du programme mais aussi des plans de tests et d'évaluation du programme. Ce plan de tests regroupe tous les tests prévus et décrit la stratégie de test. Dans ce projet nous ne vous demandons pas de rédiger un plan de tests mais nous vous demandons de bien tester votre application malgré tout. Vos tests devront être rendus en même temps que votre code source et votre exécutable.

1.1 Pourquoi faire des tests ?

Il est important de tester une application à différents niveaux (unitaire, intégration, ...). Une erreur détectée tôt dans la conception de l'application est plus facile à corriger et d'un point de vue industriel coûte moins chère.

1.2 Différents types de tests

Il existe plusieurs catégorisations des tests. En particulier, 4 grandes phases de tests sont définies :

- tests unitaires,
- tests d'intégration,
- tests systèmes,
- tests de recette et d'exploitation.

Pour chacune des phases, il existe plusieurs types de tests (exemples : test de charge, de fonctionnalité, ...). Il est à noter que l'ensemble des tests doit être rejoué après une modification du code afin de vérifier qu'il n'y a pas de régressions (l'apparition de nouvelles défaillances). Une analyse d'impact des modifications sur les cas de test peut permettre de cibler les cas de test à rejouer.

La liste des tests présentés dans ce document n'est pas exhaustive.

Tests unitaires

Un test unitaire permet de vérifier le bon fonctionnement, par rapport aux spécifications, d'une partie déterminée d'un logiciel ou d'une portion d'un programme (« unité »).

Dans ce type de test on vérifie que pour une entrée donnée, la sortie rendue par l'unité testée correspond bien à celle attendue dans les spécifications. Si c'est le cas, le test est dit "réussi", sinon le test est dit en "échec".

Tests d'intégration

Un test d'intégration est un test qui se déroule dans une phase du projet qui suit les tests unitaires. Une fois que chacun des développeurs a bien vérifié que sa partie se comporte correctement, il faut vérifier que les différentes unités se comportent bien les unes avec les autres lorsqu'elles sont assemblées ensemble.

Tests système

Les tests système viennent ensuite et vont permettre de tester l'application une fois qu'elle est complètement assemblée.

Tests de recette et d'exploitation

Le test de recette permet au client et aux intervenants d'un projet informatique de vérifier que le produit livré est bien celui qui a été commandé. Les tests de recette doivent confirmer que l'application répond correctement aux spécifications du cahier des charges. Ce ne sont pas les développeurs qui doivent effectuer ces tests, mais un utilisateur réel du logiciel. Vous n'avez donc pas à vous préoccuper de cette phase pour le projet.

2 Conclusion du rappel

Il est rare de pouvoir tester exhaustivement une application. Le nombre de configurations différentes croit de facon exponentielle. Ainsi ce n'est pas parce que tous les tests réussissent que l'application ne contient pas de "bug". Toutefois il est important de mettre en œuvre une stratégie de test pertinente, mettant en évidence le maximum d'éventuels "bugs".

Si l'on veut des assurances plus fortes de bon fonctionnement, on peut utiliser des méthodes formelles (voir le cours de Vérification de Mireille Ducassé). Il existe aussi dans le domaine du test, des méthodes, que nous ne vous demandons pas de mettre en œuvre pour ce projet, qui permettent de mesurer la couverture de code des cas de test (voir le cours de Test d'Arnaud Gotlieb).

3 Campagne de test sous Visual Studio 2010

Visual Studio vous offre la possibilité de créer un projet de test vous permettant de tester votre application. Microsoft propose une documentation appelée "Livre Blanc". Cette documentation est particulièrement utile pour des programmes écrits en C#. Pour le cas de projets écrits en C++, nous vous fournissons une documentation dans le répertoire `Ressources Projet` > `Test` > `Documentation_Test.pdf` sur le Moodle.