|  |  |
| --- | --- |
| Projet Angélique – Rapport des spécifications fonctionnelles  **Institut National des Sciences Appliquées de Rennes**  **Département Informatique**  **4ième année**  Logo_Angelique.PNG  **Octobre 2010** | |
| **Rapport de Conception Canon Noir** | |
| **Anne-Laure Berrée**  **Nolwenn Poirier** | |
|  | |
|  |
|  | |

**Encadrant : Gabriel CIRIO**

**Groupe 2.1**

|  |
| --- |
|  |

Table des matières

[Table des matières 2](#_Toc278488503)

[Introduction 3](#_Toc278488504)

[1 Spécifications générales 4](#_Toc278488505)

[2 Organisation générale de l’application 4](#_Toc278488506)

Introduction

Le projet Canon Noir a pour objectif de développer une application permettant de jouer au jeu de société Canon Noir. Ce projet nous permet de mettre en pratique les connaissances nouvellement acquises à la fois en programmation et modélisation orientée objet et en Design Pattern. Dans ce premier rapport, nous allons décrire les fonctionnalités de Canon Noir.

Cette application doit fournir à l’utilisateur une interface graphique lui permettant de jouer simplement et de façon ludique au jeu Canon Noir. Nous allons ensuite décrire plus en détail la modélisation et la conception de notre application. ?????????????

1. Spécifications générales

Les règles du jeu de Canon Noir sont assez complexes, elles sont décrites dans le sujet du projet. Nous ne reprendrons donc pas l’explication du jeu. Nous nous limitons donc au diagramme des cas d’utilisation pour décrire les différentes fonctionnalités associées à ce projet.

DIAGRAMME DE CAS D’UTILISATION

1. Organisation générale de l’application

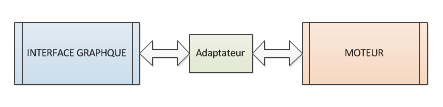
Le projet Canon se sépare en deux parties principales :

* Interface graphique
* Moteur

L’interface graphique sera en charge d’afficher le jeu et d’envoyer les différentes actions des utilisateurs au moteur du jeu. Ce module utilisera WPF.

Le moteur du jeu quant à lui a pour rôle de gérer les actions de l’utilisateur, les transitions des états du jeu, de stocker toutes les informations relatives au plateau, aux joueurs, aux dés, aux bateaux. Ce module sera codé en C++.

Ces deux modules doivent donc communiquer entre eux : l’interface doit demander au moteur ce qu’elle doit afficher, de même, elle doit lui renseigner les différentes actions de l’utilisateur. Cependant, ces deux modules ne font intervenir le même langage de programmation (C# pour l’interface et C++ pour le moteur).Il est donc nécessaire de mettre en place un Adaptateur entre ces deux modules. Cette solution a également pour avantage de ne pas faire dépendre le moteur de l’interface. Cela laisse la possibilité à une tierce personne de développer un autre modèle d’interface (par exemple : console…) sans avoir à modifier le moteur du jeu. Il devra alors implémenter son propre adaptateur. Voici le schéma représentant cette séparation des modules :



Organisation du moteur :

Le Moteur a accès aux classes :

* InfoJoueurs : classe regroupant toutes les infos concernant les joueurs
* InfoCases : classe regroupant toutes les infos concernant les cases du jeu
* InfoDes : classe regroupant toutes les infos concernant les dés

Dans la partie Moteur nous avons mis en place le DesignPattern State afin de gérer l’enchainement des différents états du jeu.

1. Diagrammes d’états-transitions associés aux états du Moteur
   1. Initialisation de la partie



* 1. Déplacement des bateaux



* 1. Tir au canon

