UFR IM²AG

UNIVERSITÉ Grenoble Alpes

Rapport de projet MÉTHODES ET OUTIL POUR LA CONCEPTION AVANCÉE

Puissance$_4$

GROUPE:6

MOUSS Adel
BELAID Mohamed
TAGUI Amine
ZITOUNI Hamza

Encadrants

LAURENCE PIERRE

LYDIE DU-BOUSQUET

LAURENT MOUNIER

Table des matières

Introduction :

Ce document a pour but de décrire le déroulement de notre projet de L'UE
MÉTHODES ET OUTILS POUR LA CONCEPTION AVANCÉE
Ce Projet Informatique porte sur un Logiciel déjà existant «Puissance_4» pour atteindre l'objectif fixé dans le cadre de la formation nous devions répondre à une succession définie par les points suivants:

- Introduction, lancement du projet (semaine 1).
- Modularité, maintenabilité, réutilisabilité (semaines 2 et 3).
- Qualité des tests, analyse de couverture (semaine 4).
- Tests pour l'analyse de vulnérabilité (semaine 5).
- Détection des défauts, correction de bugs (semaines 6 et 7).
- Résumé comparatif sur les méthodes pour le debug et pour l'analyse de vulnérabilité (semaine 8).
- Analyse de performances (semaines 9 et 10).

Ce rapport contient l'ensemble des éléments du projet:

- les spécifications plus détaillées qui en découlent.
- Nous décrirons le fonctionnement de notre projet dans son ensemble
- ainsi que les éléments qui prouvent le bon fonctionnement de celui-ci.

Spécifications fonctionnelles et détaillées

Séance 1:

Nous avons pris connaissance du programme, nous l'avons compilé à l'aide de la commande «gcc -Wall -Werror -g -o appli appli.c» pour corriger les erreurs mais aussi les warnings que nous avons observés.

Les warnings:

- warning: unused variable 'g'(supprimer la variable g)
- warning: control reaches end of non-void function
  (ajouter un return 0 à la fin de la fonction)

Les erreurs de type «Segmentation fault»

- Problème de réallocation de mémoire pour la table de jeux
- Problème de réallocation de mémoire pour la sauvegarde d'une partie
  les erreurs liées à la lecture et écriture dans les fichiers.

Séance 2:

Il convenait aujourd'hui principalement à diviser le programme en sous fichiers, chaque fichier contient des fonctions qui ont le même fonctionnement ou la même tâche :

- **appli.c** : Contient le programme principal (le « main »).
- **Initialisation.c** : Contient les fonctions d'allocation et d'initialisation.
- **Score.c** : Contient les fonctions qui calculent le score.
- **Check.c** : Contient les fonctions qui vérifient si la case choisie par le joueur est valide.
- **Iaplayer** : Contient les différents niveaux de jeux et les fonctions qui sont chargées à générer le déplacement de l'ordinateur si l'utilisateur choisi le mode solo.
- **GameBoard** : Contient les fonctions pour enregistrer le stage ou d'annuler/refaire un coup.
- **Mode.c** : Contient les fonctions des différents modes de jeux.
- **Messages.c** : Contient la fonction d'affichage des scores.

La création du Makefile:

```
CC=gcc
CFLAGS=-Wall -Werror -g
ifdef N
	CPPFLAGS=-DN=$(N)
endif
SRCS=$(wildcard *.c)
OBJS=$(SRCS:.c=.o)
EXEC=appli

all:$(EXEC)

$(EXEC): $(OBJS)
	$(CC) $(OBJS) -o $(EXEC)

.PHONY:clean exec
clean:
	-rm $(OBJS)
exec: clean
	-rm	$(EXEC)
```

► Le Makefile a été fait pour qu'il soit le plus générique possible, pour cela, on avait besoin de déclarer les variables (cf. en dessous) avec des noms précis pour qu'elles soient reconnues par la règle implicite du Makefile.

Variables utilisées :

-CFLAGS : qui contient les Flags du langage « C ».

-CPPFLAGS : nécessaire pour la déclaration d'une macro.

-SRCS : pour récupérer les noms des fichiers « .c ».

-OBJS : qui sera utilisé pour la création des fichiers objets.

-EXEC : contient le nom de l'exécutable.

► L'option « –D » a été rajoutée pour l'unique raison de récupérer une valeur « N » au moment de l'appel du Makefile, à condition que cette option à bien été entrée dans la ligne de commande ainsi que la valeur voulue, et pour vérifier cela, on s'est servi de « ifdef ».

Séance 3:

Cette séance est portée sur la création des bibliothèques et le regroupement des fichiers dans des répertoires selon leurs extensions.
bin: appli.

Header:
- appli.h
- Initialisation.h
- Score.h
- Check,h
- Iaplayer.h
- GameBoard.h
- Mode.h
- Messages.h

SRC\OBJETS:
- appli.c
- Initialisation.c
- Score.c
- Check.c
- Iaplayer.c
- GameBoard.c
- Mode.c
- Messages.c

LIB :
- LibMessages.so

Modification du Makefile :

```
CC=gcc
CFLAGS=-Wall -Werror -IHeader -ILibMessages/Header
LDFLAGS=-L LibMessages/lib -l LibMessages
ifdef L
CPPFLAGS=-DL=$(L)
endif
ifdef C
CPPFLAGS=-DC=$(C)
endif

SRCS=$(wildcard src/*.c)

OBJS=$(SRCS:.c=.o)
EXEC=bin/appli
DOXYGENDIR = Doxygen

all:$(EXEC)

doxygen:
	doxygen ./Doxyfile

$(EXEC): $(OBJS)
	$(CC) $(OBJS) -o $(EXEC) $(LDFLAGS)

.PHONY:clean,doxygen
clean:
      -rm -rf $(OBJS) $(EXEC) $(DOXYGENDIR)
```

► Ajout de l'outil de documentation automatique « doxygen » pour générer un fichier html.

► Ajout de la variable « LDFLAGS » pour inclure la bibliothèque créée.

Séance 4 :

Dans le but de corriger tout en développement le code comme il le faut, nous allons, dans cette séance, s'intéresser aux tests.
Nous allons utiliser les outils gcov et lcov pour être en capacité d'obtenir la couverture exacte de nos tests. En effet, pour que nos tests soient fiables, il faut qu'ils recouvrent idéalement la totalité des possibilités. Dans la pratique, en étant face à un nombre potentiellement infinis de possibilités, nous voudrons plutôt qu'ils couvrent un maximum de cas.
C'est pour cela que nous allons utiliser gcov et lvoc, qui nous donnent tous les deux la couverture de notre code.

Exercice 4 :

Augmenter la couverture de nos tests nous a poussé, d'une part, naturellement, à tester plus de possibilités, qui implique d'avoir un code plus fiable à l'issu de ces derniers. D'autre part, en s'ouvrant à des possibilités que nous n'avions pas prévu ou mal anticipé, nous pouvons revenir sur certains aspects du code et l'optimiser.

Exercice 6 :

Tests en cours de construction.

Modification du Makefile :

```
CC=gcc
CFLAGS=-Wall -Werror  -IHeader -IlibMessages/Header -fprofile-arcs -ftest-coverage
LDFLAGS=-L libMessages/lib -lmessages
ifdef L
CPPFLAGS=-DL=$(L)
endif
ifdef C
CPPFLAGS=-DC=$(C)
endif

SRCS=$(wildcard src/*.c)

OBJS=$(SRCS:.c=.o)
EXEC=bin/appli
DOXYGENDIR = Doxygen
GCNO=src/*.gcno
GCDA=src/*.gcda

all:$(EXEC)

doxygen:
	doxygen ./Doxyfile

$(EXEC): $(OBJS)
	$(CC) $(OBJS) -o $(EXEC) -fprofile-arcs -ftest-coverage

.PHONY:clean,doxygen
clean:
	-rm -rf $(OBJS) $(EXEC) $(DOXYGENDIR) $(GCNO) $(GCDA)
```