

Rapport de projet MÉTHODES ET OUTIL POUR LA CONCEPTION AVANCÉE

Puissance₄

**GROUPE:6**

MOUSS Adel

BELAID Mohamed

TAGUI Amine

ZITOUNI Hamza

**Encadrants**

LAURENCE PIERRE

LYDIE DU-BOUSQUET

LAURENT MOUNIER

DAVID MONNIAUX

Table des matières

[Table des matières 2](#_Toc2761493)

[Introduction : 3](#_Toc2761494)

[Spécifications fonctionnelles et détaillées 3](#_Toc2761495)

[Séance 1: 3](#_Toc2761496)

[Séance 2: 4](#_Toc2761497)

[Séance 3: 5](#_Toc2761498)

[Séance 4 : 6](#_Toc2761499)

[ Exercice 4 : 6](#_Toc2761500)

[ Exercice 6 : 6](#_Toc2761501)

[Séance 5: 7](#_Toc2761502)

[ Exemple de programme contenant une erreur : 7](#_Toc2761503)

[ Résultat de l’exercice 4: 9](#_Toc2761504)

[Séance 6 : 9](#_Toc2761505)

[ L’outil KLEE : 9](#_Toc2761506)

[ L’outil valgrind : 11](#_Toc2761507)

# 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:

► 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.

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)

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: SRC\OBJETS:

appli.h appli.c

Initialisation.h Initialisation.c

Score.h Score.c

Check,h Check.c

Iaplayer.h Iaplayer.c

GameBoard.h GameBoard.c

Mode.h Mode.c

Messages.h 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.

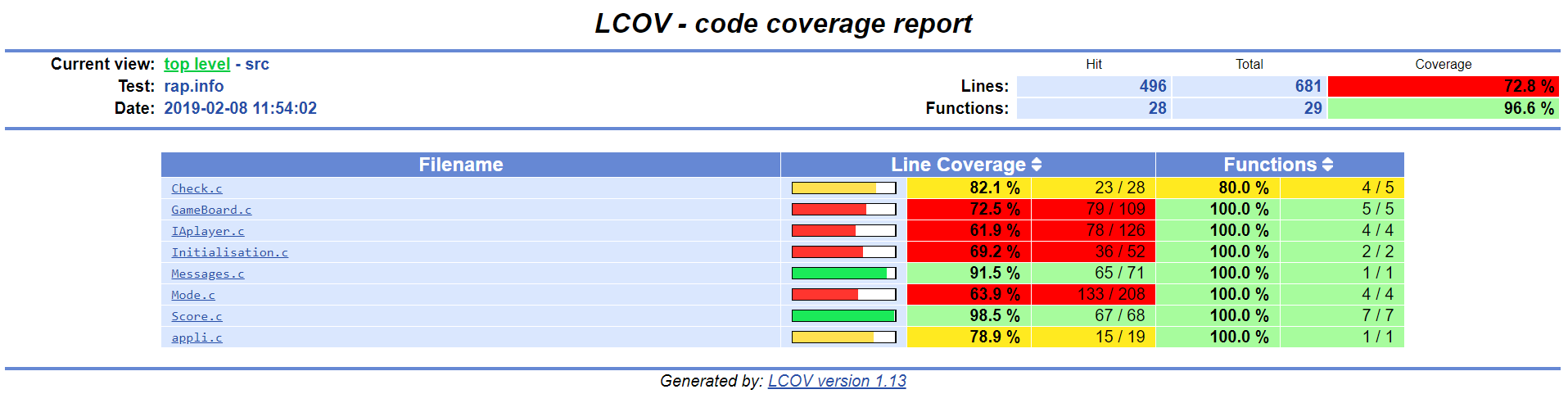
C’est pour cela que nous allons utiliser gcov et lvoc, qui nous donnent tous les deux la couverture de notre code.

### 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.

Couverture des tests :

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)

## Séance 5:

### Exemple de programme contenant une erreur :

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]){

int t[10000];

int i,c;

FILE\* f;

f = fopen(argv[1],"r");

fscanf(f,"%d",&c);

for(i=0;i<c;i++){

printf("---%d---\n",i);

t[i]=i;

}

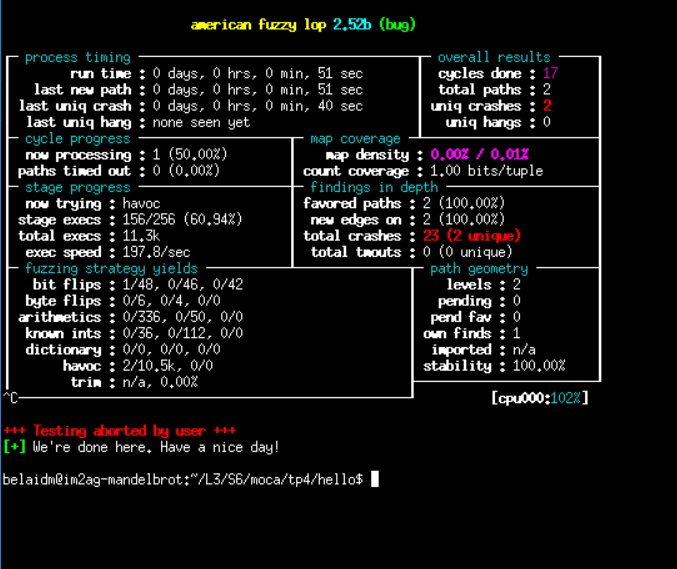
printf("t[%d]\n",t[i\*2]);

fclose(f);

return 0;

}

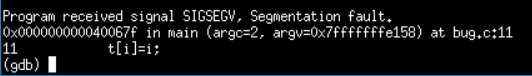
Après avoir lancé AFL et laissé le FUZZER tourner, 2 crashes **uniques** ont étaient détectés (comme l’indique la capture d’écran en dessous).



* 2 fichiers ont étaient créer par AFL pour provoquer ces crashs :

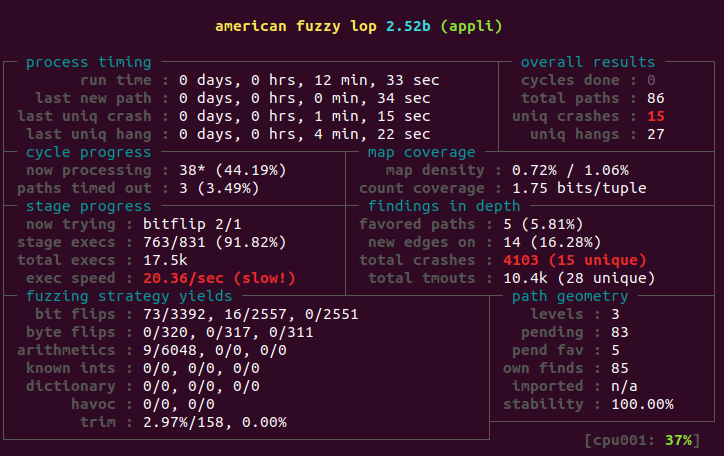
|  |  |
| --- | --- |
| Fichier | Contenu |
| id:000000,sig:11,src:000001,op:havoc,rep:64 | 11111\_€\_\_111111111:11111E1011 11O1111Q+111111111\_\_1111 |
| id:000001,sig:11,src:000001,op:havoc,rep:64 | 33333ÿÿ3333333333\_\_ÿÿ\_3333ä |

Après avoir lancé gdb avec les fichiers générés, la même erreur a été détectée pour les deux fichiers



### Résultat de l’exercice 4:

Pour un fuzzing de plus de 12 minutes, on obtient le résultat suivant :



## Séance 6 :

### L’outil KLEE :

Le but de cette séance est de corriger toutes les erreurs concernant la mémoire, pour cela, l’outil KLEE a été utilisé.

Pour lancer KLEE sur notre projet (puissance 4), il a fallu modifier tout d’abord le Makefile pour inclure la bibliothèque contenant KLEE et aussi pour rajouter l’instruction qui génère un fichier “bitcode” qui, lui-même, sera exécuté après avec la commande “klee” pour afficher les erreurs détectées par cet outil.

Ce qui a été remarqué après l’exécution du fichier “.bc”, de nombreuses erreurs ont été signalées concernant les fonctions de la librairie standard du “C”, et c’est parce-que “Klee” n’arrive pas à trouver le chemin vers les fichiers où ces fonctions ont étaient déclarées ce qui explique les messages d’erreurs affichées, pour éviter cela, il est nécessaire de remplacer les “scanf” par des “lectures symboliques” (les “printf” peuvent être remplacés par des opérations qui ne fassent rien).

Les modifications apportées sur le Makefile :

CC=wllvm

CFLAGS=-Wall -g -IHeader -ILibMessages/Header -I $(KLEE)/include

LDFLAGS=-L LibMessages/lib

LDLIBS=-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

all:$(EXEC) $(EXEC).bc

$(EXEC) : LibMessages.a

LibMessages.so: src/Messages.c

$(CC) -fPIC -c src/Messages.c

$(CC) -shared -o LibMessages/lib/LibMessages.so/Messages.o

LibMessages.a: src/Messages.o

$(AR) cr $@ $+

doxygen:

doxygen ./Doxyfile

$(EXEC): $(OBJS)

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

extract-bc $(EXEC)

.PHONY:clean,doxygen

clean:

-rm -rf $(OBJS) $(EXEC) $(DOXYGENDIR) $(EXEC).bc

### L’outil valgrind :

Pour cette partie, on n’avait pas besoin de modifier grand-chose, les seules modifications qui ont été effectuées été sur le Makefile, pour rajouter tout simplement le « -g » à la variable « CFLAGS » qui nous permettra de compiler notre programme avec l’option de débogage pour pouvoir l’exécuter ensuite avec « valgrind ».

Modification apportée sur le « Makefile » : l’ajout de l’option –g a la variable « CFLAGS » (cf ci-dessous).

CC=wllvm

CFLAGS=-Wall -g -IHeader -ILibMessages/Header -g

LDFLAGS=-L LibMessages/lib

LDLIBS=-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

all:$(EXEC) $(EXEC).bc

$(EXEC) : LibMessages.a

LibMessages.so: src/Messages.c

$(CC) -fPIC -c src/Messages.c

$(CC) -shared -o LibMessages/lib/LibMessages.so/Messages.o

LibMessages.a: src/Messages.o

$(AR) cr $@ $+

doxygen:

doxygen ./Doxyfile

$(EXEC): $(OBJS)

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

.PHONY:clean,doxygen

clean:

-rm -rf $(OBJS) $(EXEC) $(DOXYGENDIR)