****

**North South University**

**Project Report**

**CSE 427**

**Sec: 1**

Title: Tick-Tac-Toe

**Submitted By:**

|  |  |  |
| --- | --- | --- |
| Muhammad Sakib Khan | 1520016042 | muhammad.sakib@norhsouth.edu |
| Group:10 | | |

GitHub Project Link:

<https://github.com/nsuspring2019cse427/Group10>

**Submitted to:**

**Shaikh Shawon Arefin Shimon**

Lecturer,

Department of Electrical and Computer Engineering

North South University

Table of Contents

[Project Description 3](#_Toc7492462)

[Introduction 3](#_Toc7492463)

[Testing Aspects 3](#_Toc7492464)

[Tools/Frameworks Used: 3](#_Toc7492465)

[Description of Input Space Partitioning 4](#_Toc7492466)

[Class: GameEngine.java. (ISP done in GameEngineTest.java) 4](#_Toc7492467)

[Method: play (int x, int y) 4](#_Toc7492468)

[Method: elt (int x, int y) 4](#_Toc7492469)

[Graph Partitioning 5](#_Toc7492470)

[Class: GameEngine.java 5](#_Toc7492471)

[Methods: 5](#_Toc7492472)

[Class: MainActivity.java 11](#_Toc7492473)

[Methods: 11](#_Toc7492474)

# Project Description

## Introduction

Classic games that were once made for desktop computers at first are now commonly developed for smartphones. Android Operating System is highly used in modern smartphones. In this project, testing of one of such classic games on Android known as Tic-Tac-Toe has been done. The application has been developed using Java Programming Language in backend hence testing was done in Java Programming Language.

The project that has been tested by me was initially developed and available publically at: <https://www.ssaurel.com/blog/learn-to-create-a-tic-tac-toe-game-for-android/>

## Testing Aspects

The following testing aspects have been implemented:

* Unit Testing methods
* Input Space Partitioning
* Graph Partitioning
* Integration Testing
* UI Testing

## Tools/Frameworks Used:

* Android Studio
* JUnit4
* Espresso (For UI Testing)

# Description of Input Space Partitioning

Interface based approach has been used to do input space partitioning.

The Tic-Tac-Toe grid is a 3X3 grid starting from (0, 0) till (2, 2). Hence negative values are considered as the equivalent positive as the negative sign is omitted.

## Class: GameEngine.java. (ISP done in GameEngineTest.java)

### Method: play (int x, int y)

|  |  |  |
| --- | --- | --- |
| Input Characteristics | Values | Code Line in Test Class |
| Both Inputs are positive and same | (1,1) | 162-176 |
| Both inputs are positive but not same | (2,1) | 145-159 |
| Both inputs are zero (Min. Grid Position) | (0,0) | 127-141 |
| Both Inputs are negative | (-2,-1) | 180-198 |
| One Input is negative another is positive | (-2,1) | 202-220 |
| Both inputs are same (Max. Grid Position) | (2,2) | 108-122 |

Table 1: Partition for play(int x, int y)

### Method: elt (int x, int y)

|  |  |  |
| --- | --- | --- |
| Input Characteristics | Values | Code Line in Test Class |
| Both inputs are zero (Min. Grid Position) | (0,0) | 238-249 |
| Both inputs are positive but not same | (1,2) | 252-263 |
| Both Inputs are negative | (-2,-1) | 268-282 |
| Both Inputs are negative same (Neg. Max. Grid Position) | (-2,-2) | 285-299 |
| Both inputs are same (Max. Grid Position) | (2,2) | 224-235 |

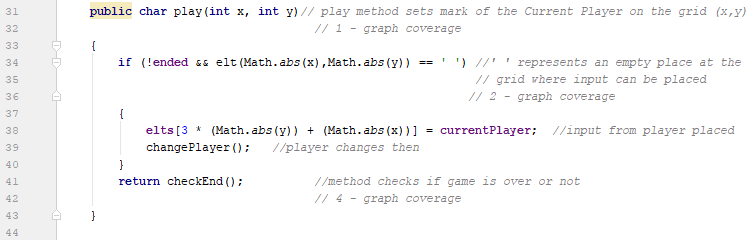
Table 2: Partition for elt (int x, int y)

# Graph Partitioning

## Class: GameEngine.java

### Methods:

#### play (int x, int y)



C

B

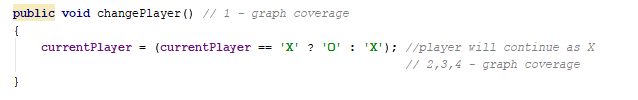
A

A !A

B

C

#### changePlayer ( )



C

A

B

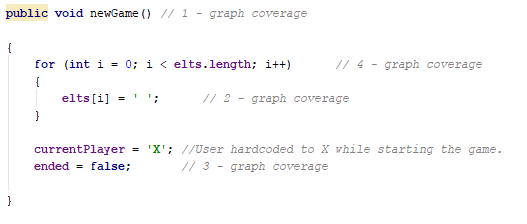
A B

C

C

Hardcoded to X

#### newGame ( )



D

B

C

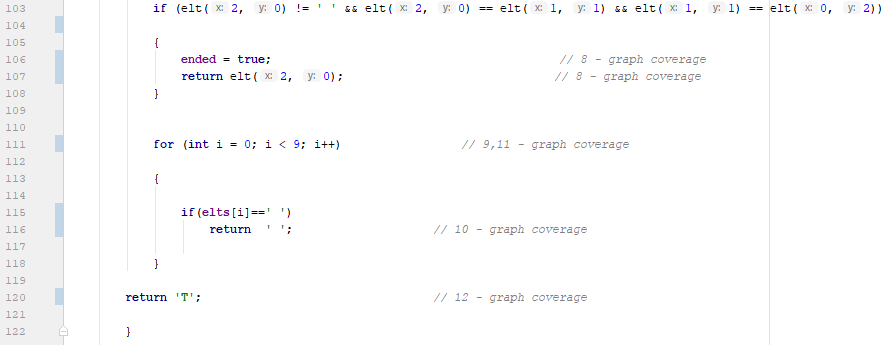
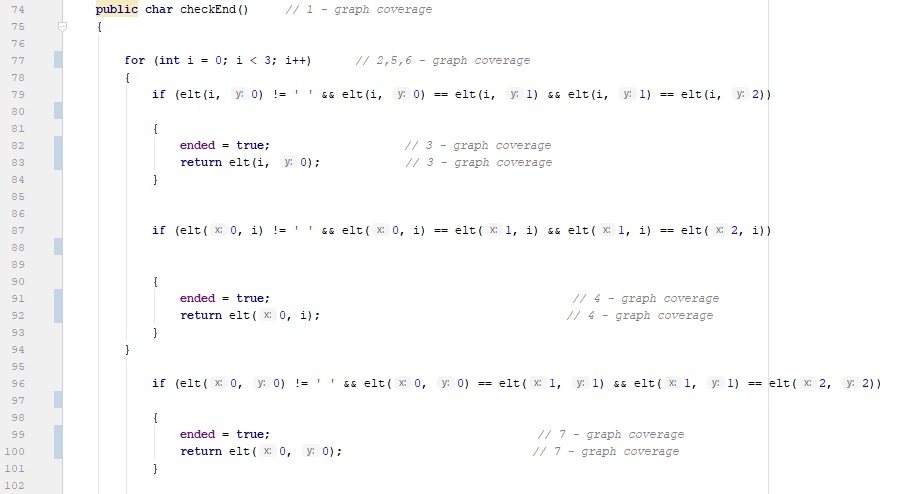
A

A !A

B D

C

#### checkEnd ( )



P

O

N

M

L

K

A

J

I

H

G

F

E

D

C

B

A H J L

B E I K M !M

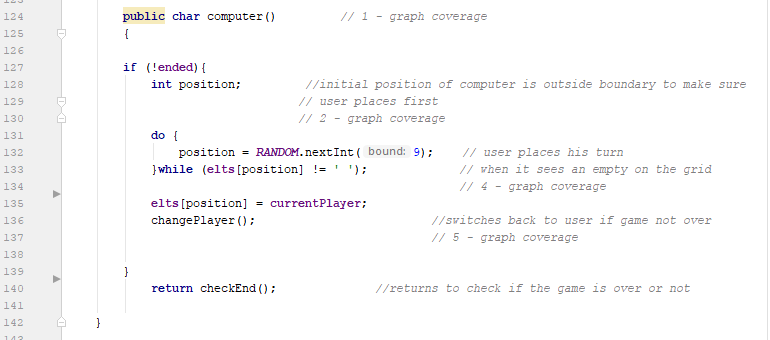
C F

N P

D G

O

#### computer ( )



D

F

E

C

B

A

A !A

B D

C D

F

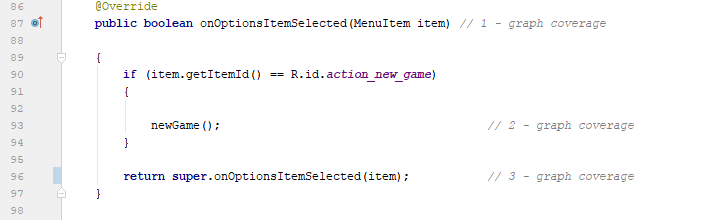
D!

E

## Class: MainActivity.java

### Methods:

#### onOptionsItemSelected (MainItem item)



C

B

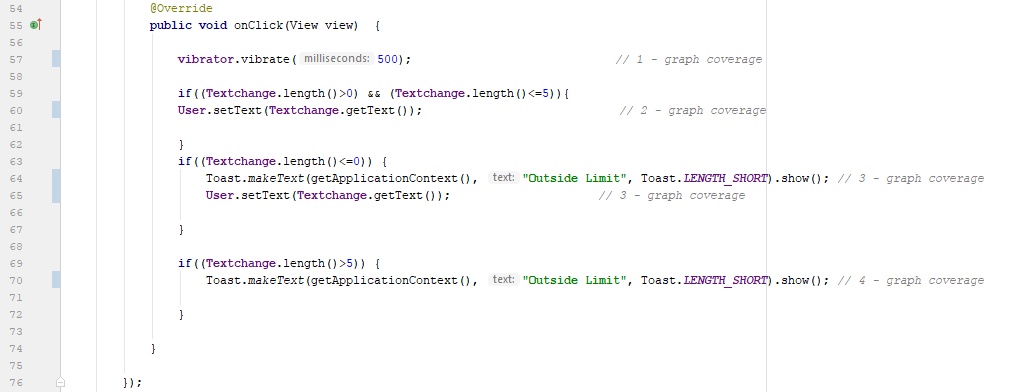
A

A !A

B C

#### 

#### OnClick (View view)



F

G

D

A

E

C

B

A

B D F

C G  
 E

Node Coverage and Edge Coverage = [1, 2], [1, 3], [1, 4]