Testing Go Simulation Program

Course Project of CSC4001 Software Engineering 2025 Spring

Qingshuo Guo
qingshuoguo@link.cuhk.edu.cn

Qiuyang Mang
qiuyangmang@link.cuhk.edu.cn

1 INTRODUCTION

In this project, you need to test a Go simulation program written in Python. This project covers two main topics: **(a)** *Differential Testing* (50 pts) and **(b)** *Metamorphic Testing* (50 pts). You are required to implement code (80 pts) and write only **one** report (20 pts) for all parts. We also provide a bonus task (10 + 10 pts) for both topics. You may earn extra points from the bonus task, but your final score won't exceed 100 pts.

2 PRELIMINARIES

Go is a strategy board game from China. It is played on a 19×19 grid. Two players take turns placing stones: one plays black, and the other plays white. **In this project, we will use modified Go rules, which differ slightly from the standard Go rules.**

2.1 Overview

The game of Go follows a structured sequence of play, which can be described as follows:

(1) **Board Setup**: The game begins with an empty board of size 19×19.
(2) **Turn Order**: Black moves first, and players alternate turns.
(3) **Placing Stones**: On their turn, a player places a stone of their color on an empty intersection of the board.
(4) **Capturing Stones**: If a player's move removes the last liberty of some stones, those stones are captured and removed from the board. (we will explain the detailed rules of capturing stones in the next section).

2.2 Capturing Stones

Capturing occurs when a stone or a group of stones has no remaining liberties. A **liberty** is an empty intersection adjacent to a stone. The capturing process is explained as follows:

2.2.1 Single-Stone Capture. Each placed stone initially has four liberties (except for those on the edge or in a corner, which have fewer). If all of a stone's liberties are occupied, it is captured and removed from the board.

2.2.2 Group Capture. Stones of the same color that are directly connected (vertically or horizontally) form a **group**. A group shares its liberties collectively. If all liberties of a group are occupied, the entire group is captured and removed from the board.

2.2.3 Examples. Here we will show some examples of capturing stones and we will use plain text to draw the board, where '.' represents an empty intersection, 'B' represents an intersection occupied by a black stone, and 'W' represents an intersection occupied by a white stone. For example, the following text represents a board of size 3×3, with a white stone in the middle and three black stones on the top, left, and bottom of the white stone respectively, all the remaining intersections are empty.

```
. B .
B W .
. B .
```

Example1. Suppose it is the turn of the black player and before he moves, the board is

```
. B .
B W .
. B .
```

He chooses to place a black stone on the second row and third column.

```
. B .
B W B
. B .
```

Therefore, all the liberties of the white stone are occupied and it will be removed from the board.

```
. B .
B . B
. B .
```

Example2. Suppose the white stone is at the corner now.

```
W . .
B . .
. . .
```

The black player chooses to place his stone on the first row and the second column.

```
W B .
B . .
. . .
```

Similarly, all the liberties of the white stone are occupied and it will be removed from the board.

```
. B .
B . .
. . .
```

Example3. Suppose the white stones forms a group now.

```
. B B .
B W W .
. B B .
```

The black player chooses to place his stone on the second row and the fourth column.

```
. B B .
B W W B
. B B .
```

Similarly, all the liberties of the white stone group are occupied and they will be removed from the board.

```
. B B .
B . . B
. B B .
```

Example4. Suppose these two stones overlaps each other

```
. B W .
B W . W
. B W .
```

The black player chooses to place his stone on the second row and the third column.

```
. B W .
B W B W
. B W .
```

All the liberties of the white stone on the second row and second column are occupied and it is the same case for the black stone on the second row and the third column. Both two stones will be removed from the board.

```
. B W .
B . . W
. B W .
```

2.3 Go Simulation Program

The Go simulation program takes a text file as input, which contains a sequence of moves made by two players. It processes these moves and outputs a text file representing the final state of the board. For example, consider the following input file:
Input File: Each line represents a move, where the first number is the row index, the second number is the column index, and players alternate turns, starting with Black.

```
3 3
4 3
3 4
4 4
2 3
```

The program processes these moves and generates the final board state as follows:
Output File: Black stones are represented by 'B', white stones by 'W', and empty spaces by '.'.

```
. . . . . . . .
. . B . . . . .
. . B B . . . .
. . W W . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Here, the black and white players take turns one by one. The output board shows the final state after applying the sequence of moves.

2.3.1 Rules and Assumptions.

- The board is assumed to be of fixed size (e.g., 19×19 for the offical codes; for the sake of simplicity, we use 7×7 for examples in the document).
- The moves are applied sequentially in the order given in the input file.
- If one line represents an invalid move (i.e., an empty line, index of invalid intersection, or one line containing an invalid character), we assume this player skipped this turn and passed it to the opponent.

2.4 Examples

2.4.1 Example 1: Invalid move 1. **Input File:**

```
100 100
3 2
```

Output File:

```
. . . . . . . .
. . . . . . . .
. W . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

2.4.2 Example 2: Invalid Move 2. **Input File:**

```
3 3
4 3
3 4
X Y
```

Output File:

```
. . . . . . . .
. . . . . . . .
. . B B . . . .
. . W . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

2.4.3 Example 3: Invalid move 3. **Input File:**

```
2 2
2 2
```

Output:

```
. . . . . . . .
. B . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

2.4.4 Example 3: Invalid move 4. **Input File:**

```
2 2
7 7 aaa
```

Output:

```
. . . . . . . .
. B . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

2.4.5 Example 5: **Input File:**

```
4 4
3 4
1 1
5 4
1 7
4 3
7 1
4 5
```

Output File:

```
B . . . . . B
. . . . . . .
. . . W . . .
. . W . W . .
. . . W . . .
. . . . . . .
B . . . . . .
```

2.4.6 Example 6: **Input File:**

```
4 4
3 4
1 1
5 4
1 7
4 3
7 1
4 5
4 4
```

Output File:

```
B . . . . . B
. . . . . . .
. . . W . . .
. . W . W . .
. . . W . . .
. . . . . . .
B . . . . . .
```

2.4.7 More Examples. You can run the released 'simulator_correct.py' to test more examples.

3 TESTING

This section deals with GO simulation programs that hide bugs. These bugs will even yield incorrect outcomes for inputs. Your task is to employ two testing methodologies in this course, Differential Testing and Metamorphic Testing, to automatically detect bugs within each GO simulation program.

3.1 Differential Testing (50 + 10 pts)

Overview. To find bugs in a buggy Go simulation program through differential testing [1, 2], **you need to write a generator for the GO simulation program**. You are required to submit one Python file for this task: "generator_diff.py" (your generator). For each buggy interpreter, we will repeat the testing process with 100 iterations:

(1) Run your generator "generator_diff.py", which should save the program output to a file named "1.in" under the same directory.
(2) Use "1.in" as input and run the correct simulator "simulator_correct.py", then save the output to a file named "1.out" under the same directory.
(3) Use "1.in" as input again, this time with the buggy simulator, and save the output to a file named "2.out" under the same directory.
(4) Compare "1.out" and "2.out". Any inconsistency between them will serve as a bug report for your testing.

Note that your Python code should include the file operations (see the released code template).

Requirements. Your generator should output a **non-empty** text file with no more than 1,000 lines each time, and there should not be more than 1,000 characters in each line, and it should generate the text file within one second.

Grading. (**40 pts for testing on buggy simulators**) We have 10 buggy simulators in total. 5 source codes has been released, and the remaining 5 are hidden. For each simulator, the task has **4 pts** in total, you will earn this **4 pts** if your program finds a bug report with 100 iterations.
(**10 pts for report**) You will receive full grades for the report as long as you demonstrate comprehension of the core concept of differential testing and your code. So do not worry about it.
(**10 bonus pts for testing on student-submitted simulators**) Each student is allowed to submit one buggy simulator (optional). We will test each student's solutions on all the simulators submitted by students and rank them by the number of buggy simulators they detected. The bonus points will be assigned according to the following rules.

(1) (10 pts) if the number of buggy simulators you detected is among the top 10% of all students.
(2) (8 pts) if the number of buggy simulators you detected is among the 11 - 30% of all students.
(3) (6 pts) if the number of buggy simulators you detected is among the 31 - 50% of all students.
(4) (4 pts) if the number of buggy simulators you detected is among the 51 - 70% of all students.
(5) (2 pts) if the number of buggy simulators you detected is among the 71 - 90% of all students.
(6) (0 pts) if the number of buggy simulators you detected is among the 91 - 100% of all students.

Hints.

(1) You may evaluate your test case quality by testing the code coverage on the released correct simulator.

3.2 Metamorphic Testing (50 + 10 pts)

Overview. In contrast to differential testing [1, 2], we will not implement another version of the target system to detect bugs. Instead, we utilize the relationship between the system outputs of the given inputs related. To utilize metamorphic testing [3] for Go simulators, your task is to implement a generator that can produce two related inputs and a checker for their outputs, which are named "generator_meta.py" and "checker.py" respectively.
For each buggy simulator, we'll repeat the testing process with 100 iterations.

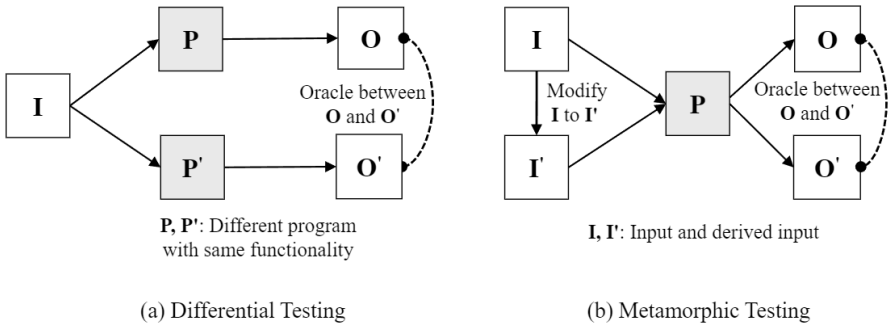

Figure 1: Two testing methods

(1) Run your generator "generator_meta.py", which should save two files named "1.in" and "2.in".
(2) Use "1.in" as input and run the buggy simulator, then save the output to a file named "1.out".
(3) Use "2.in" as input and run the buggy simulator, then save the output to a file named "2.out".
(4) Run your checker, which should read "1.out" and "2.out" and then write a single integer in $\{0, 1\}$ to the file "res.out", where 1 indicates the buggy simulator output at least one incorrect result for the two inputs.

Note that your Python code should include the file operations (see the released example).

Requirements. Similar to Section 3.1, your generator should output two text files with no more than 1,000 lines each time, and there should not be more than 1,000 characters in each line and it should generate the text files within one second. In this part, we only further consider the successful iteration when your checker reports "1". There are two states:

(1) **TP**: Any of the two results of the buggy simulator is different from the correct simulator when the checker reports "1". It means you successfully find a bug-inducing test case. Congratulations!
(2) **FP**: All results of the buggy simulator are the same as the correct version when the checker reports "1".

Grading. (**40 pts for testing on buggy simulators**) We have 10 buggy simulators in total. 5 source codes has been released, and the remaining 5 are hidden. For each simulator, the task has **4 pts** in total. You will earn:

$$4 \times \frac{\#\text{ of TP reports}}{\max(\#\text{ of TP reports} + \#\text{ of FP reports}, 1)}\ \text{pts.}$$

(**10 pts for report**) You will receive full grades for the report as long as you demonstrate comprehension of the core concept of metamorphic testing and your code. So do not worry about it.
(**10 bonus pts for testing on student-submitted simulators**) Each student is allowed to submit one buggy simulator (optional). We will test each student's solution on all the simulators submitted by students and rank them by the sum of the following scores on each simulator.

$$\frac{\#\text{ of TP reports}}{\max(\#\text{ of TP reports} + \#\text{ of FP reports}, 1)}\ \text{pts.}$$

The bonus points will be assigned according to the following rules.

(1) (10 pts) if your score is among the top 10% of all students.
(2) (8 pts) if your score is among the 11 - 30% of all students.
(3) (6 pts) if your score is among the 31 - 50% of all students.
(4) (4 pts) if your score is among the 51 - 70% of all students.
(5) (2 pts) if your score is among the 71 - 90% of all students.
(6) (0 pts) if your score is among the 91 - 100% of all students.

Hints.

(1) A possible metamorphic relation (may not sound) is to reverse the movement sequence of two players (i.e., making the first move an invalid move, after which the order will be reversed).
(2) You may need to combine multiple metamorphic relations to enhance the testing performance.

3.3 Notes

- Your Python code should avoid importing any third-party libraries, which may cause you to get **0 pts** for the corresponding sub-question.
- Any disallowed file operations may result in a score of **0 pts** for the whole project.
- For the metamorphic testing part, invalid actions that contradict the principles of metamorphic testing, such as storing the correct output within the checker for direct comparison, may result in a score of **0 pts** in this part.
- The bug you introduce should be reasonable for the buggy program submitted by the student. We may discard your buggy simulator in the final testing if the bug is too specific (e.g., it is triggered only by a particular pattern on the board) or too similar to bugs introduced by other students. The bug should not cause the program to crash, hang indefinitely, or produce an error message, and your program should output within a reasonable time for any valid input.
- The hidden buggy simulator will be similar to the released buggy simulator, with some of them having bugs that are more subtle than those in the released version. You can check the code coverage of your generator to ensure that it triggers the

bug in the hidden buggy simulator. Additionally, you can create your buggy simulators to evaluate your generator.

4 MATERIALS

We will release the following materials and you can use those materials to evaluate your grade for this project.

- "simulator_correct.py" – correct version of the simulator (**do not modify the content of this code**).
- "simulator_buggy" – the folder contains buggy versions of the simulator (**do not modify the content in this folder**).
- "grader.py" – code for yourself to evaluate your grade on the released buggy simulator (**do not modify the content of this code**).
- "generator_diff.py" – implement your generator for differential testing.
- "generator_meta.py" – implement your generator for metamorphic testing.
- "checker.py" – implement your checker for metamorphic testing.

5 SUBMISSION

The deadline for this project is April 28th. Late submissions will receive a grade of 0 for the whole project. Your submission should include the following 4-5 files named as follows. Please submit them in a single submission without zipping through Blackboard. All code grading will be under Ubuntu20.04 and Python3.9 basic environment.

- "generator_diff.py" your generator to generate movement sequences for differential testing.
- "generator_meta.py" your generator to generate movement sequences for metamorphic testing.
- "checker.py" your checker for metamorphic testing.
- (optional) "simulator_buggy.py" your implementation for buggy simulator.
- "report.pdf" your report for the whole project.

Please note that, TAs may ask you to explain the meaning of your program, to ensure that the codes are indeed written by yourself. Please also note that we would check whether your program is similar to your peers' code using plagiarism detectors. If you have any questions regarding this project, please send an email to TAs or the USTFs[1] for this project.

6 Q&A SESSION FOR THIS PROJECT

We will conduct Q&A sessions during the week of April 7th to April 13th. Please feel free to ask your questions at that time.

[1] qingshuoguo@link.cuhk.edu.cn and qiuyangmang@link.cuhk.edu.cn

REFERENCES

[1] Robert B Evans and Alberto Savoia. 2007. Differential testing: a new approach to change detection. In *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*. 549–552.

[2] William M McKeeman. 1998. Differential testing for software. *Digital Technical Journal* 10, 1 (1998), 100–107.

[3] Sergio Segura, Gordon Fraser, Ana B Sanchez, and Antonio Ruiz-Cortés. 2016. A survey on metamorphic testing. *IEEE Transactions on software engineering* 42, 9 (2016), 805–824.