Lecture 11 — Automated Testing I

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

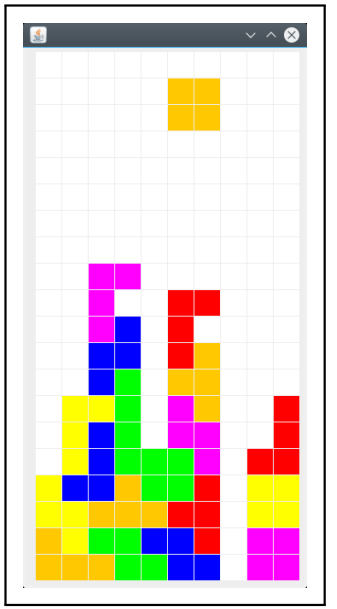
Random Testing

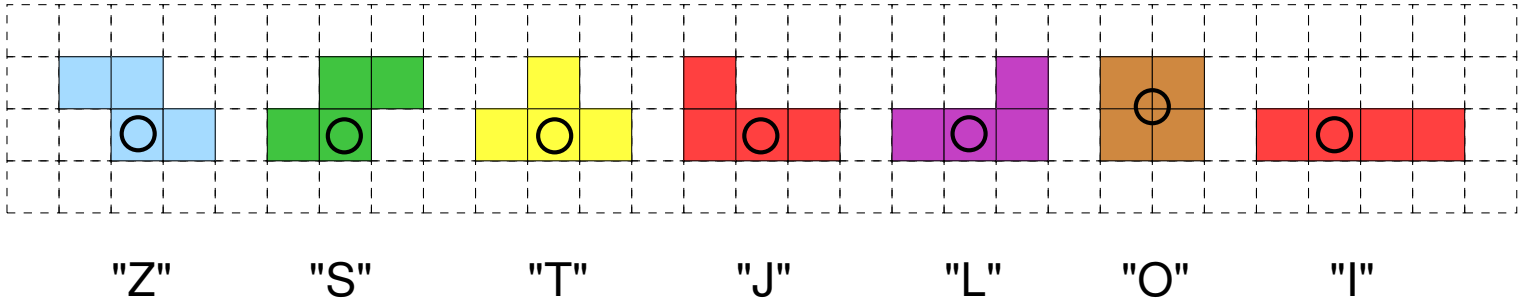
"*Random testing is a black-box software testing technique where programs are tested by generating* **random, independent inputs**. *Results of the output are compared against software specifications to verify that the test output is pass or fail. In case of* **absence of specifications** *the exceptions of the language are used which means if an exception arises during test execution then it means there is a fault in the program, it is also used as way to avoid biased testing.*"
–Wikipedia

- Randomised testing is a **powerful** but **challenging** idea!
- Why write tests when they can be **automatically generated?**

Tetris: Overview

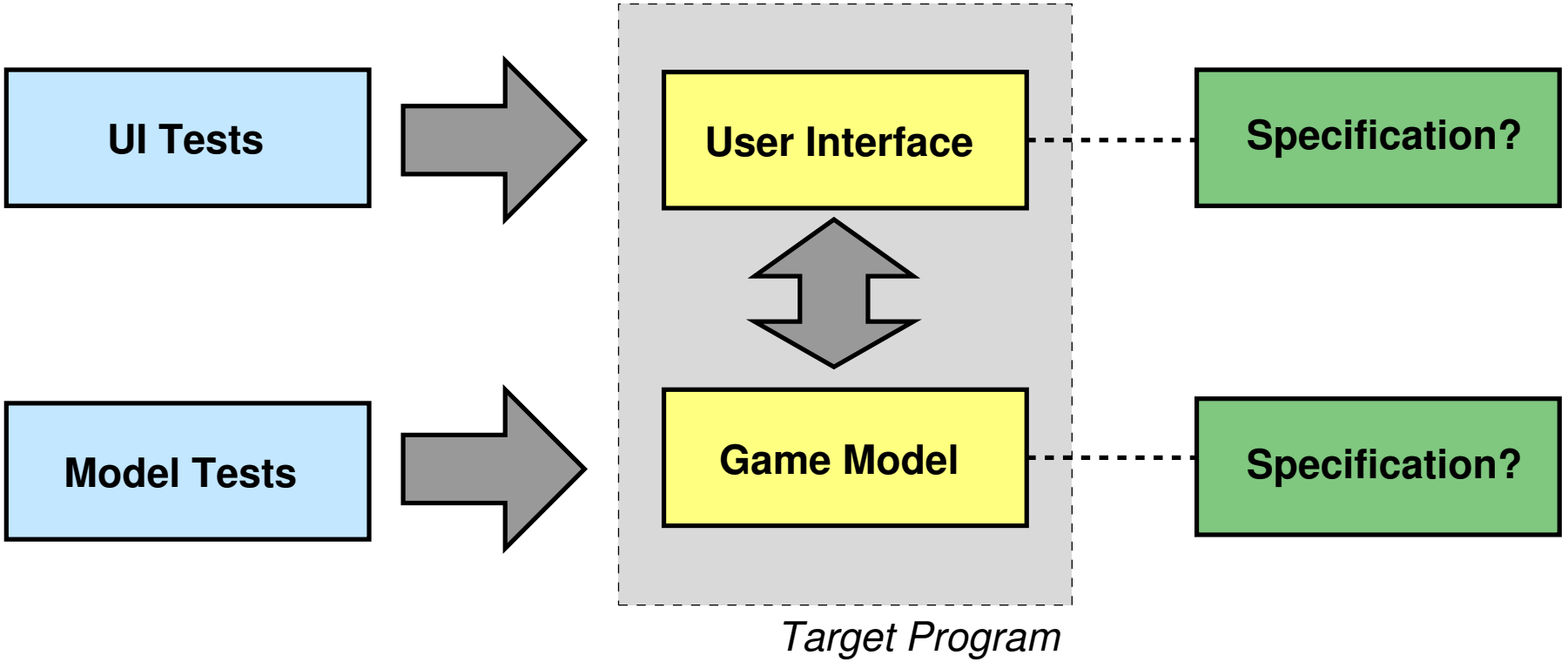
- **Tetromino's** controlled using keyboard
- Tetromino's move **left** or **right**, and **rotate**
- Rows which become full are **removed**





Question: *How to randomly test this program?*

Tetris: Testing Architecture



- **Q)** *What to test?*
- **Q)** *What is it supposed to do??*

Tetris: Test Sequences

```
(T blue; drop; drop; drop; land)
(S red; rotate; right; drop; drop; land)
(Z magenta; rotate; left; drop; land)
```

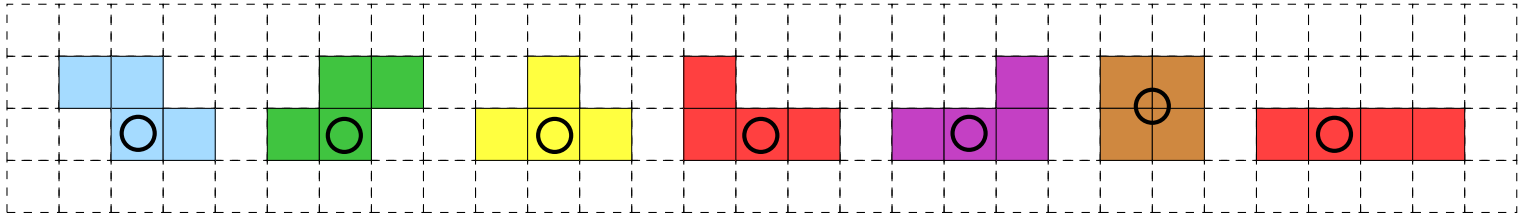
- Textual notation allows **decoupling** from model

- For 5x5 Tetris board, above sequence should generate this

```
|_|_|_|_|_|
|_|_|M|_|_|
|_|M|M|R|_|
|_|M|B|R|R|
|_|B|B|B|R|
```

- Again, textual notation to **decouple** user interface

Tetris: Random Test Sequences



- **Tetrominos:** `Z`, `S`, `T`, `J`, `L`, `O`, `I`
- **Colors:** `Blue`, `Green`, `Yellow`, `Red`, `Magenta`, `Brown`
- **Moves:** `Drop`, `Left`, `Right`, `Rotate`
- **Events:** *Wall Collision*, *Piece Collision*, *Landing*, *Next Piece*, *Line Removal*, *Quadline Removal*, *Game Over*

Tetris: Sequence Depth

- *Probability of generating "***rotation***" move?*
- *Probability of generating "***Landing***" event?*
- *Probability of generating "***Line Removal***" event?*
- *Probability of generating "***Quadline Removal***" event?*
- *Probability of generating "***Game Over***" event?*

Q) *How to ensure tests generated for all events?*

Tetris: Example Data

- Randomly generating **100** games ...

  ... with 3 x **turns**, 2 – 4 **moves** per turn

  ... yielded 909 **individual moves**

  ... with 213 **landing events**

  ... and 3 **single line removals**

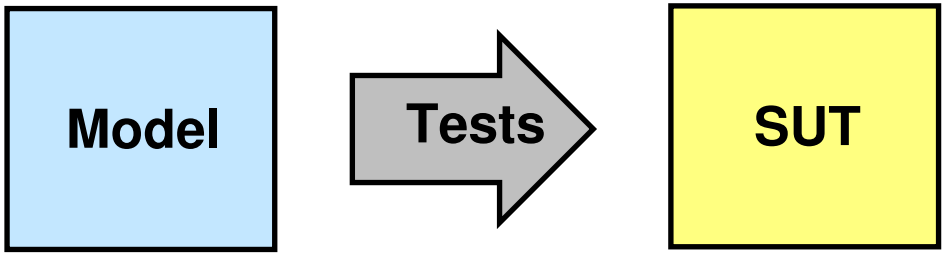
  ... and 0 **other line removals**

Tetris: Test Oracle

"*A* **test oracle***, or just oracle, is a mechanism for determining whether a test has passed or failed. The use of oracles involves comparing the output(s) of the system under test, for a given test-case input, to the output(s) that the oracle determines that product should have.*"
–Wikpedia

- *For SWEN221 assignment, what "oracle" might we use?*
- *What are the problems with this "oracle"?*

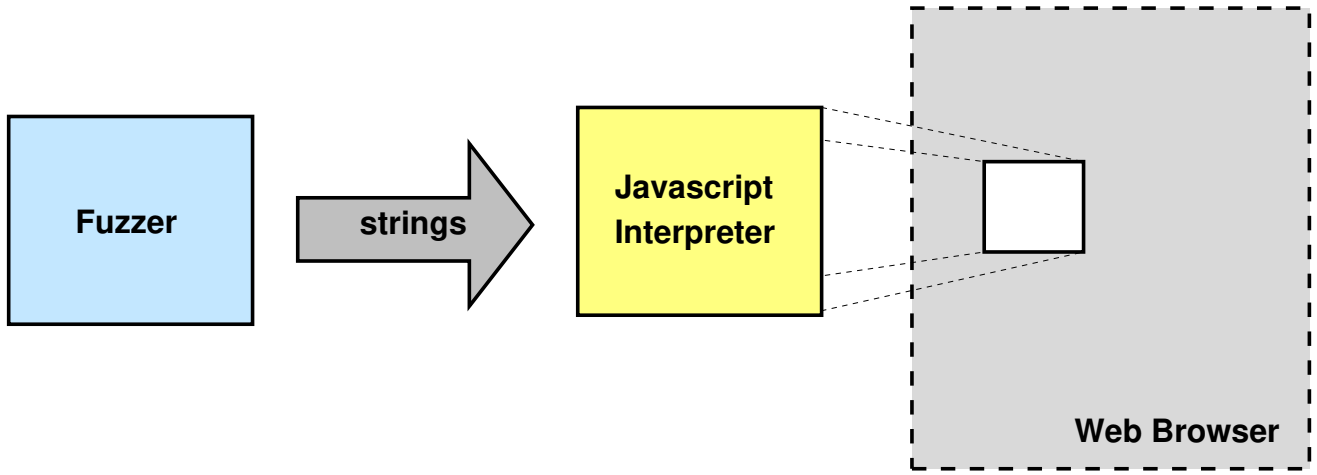
Model-Based Testing



- Test cases generated from model of **System Under Test (SUT)**
- **Model** should be simpler than SUT
- Ideally, want easy way to know model is **correct**
- Model **abstracts** system in different ways (e.g. to focus on something)
- Model provides **specification** for aspect it tests

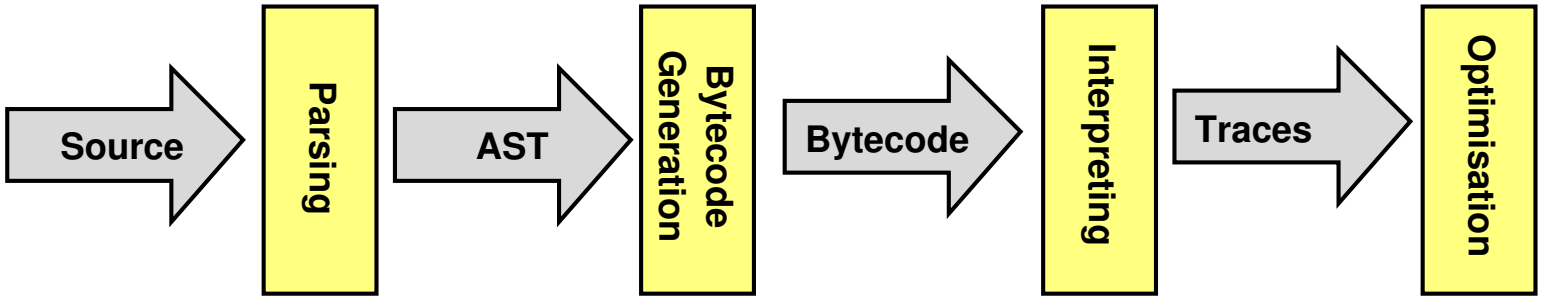
JavaScript Interpreter: Case Study

"*Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as* **inputs** *to a computer program. The program is then monitored for* **exceptions** *such as crashes, or failing built-in code assertions or for finding potential memory leaks.* "



- *Why is this important? What are the challenges?*

JavaScript Interpreter: Case Study




```
function sum(arr) {
  var sum = 0;
  for(var i=0;i!=arr.length;++i) {
    sum = sum + arr[i];
  }
  return sum;
}
```

- *What are we trying to test?*

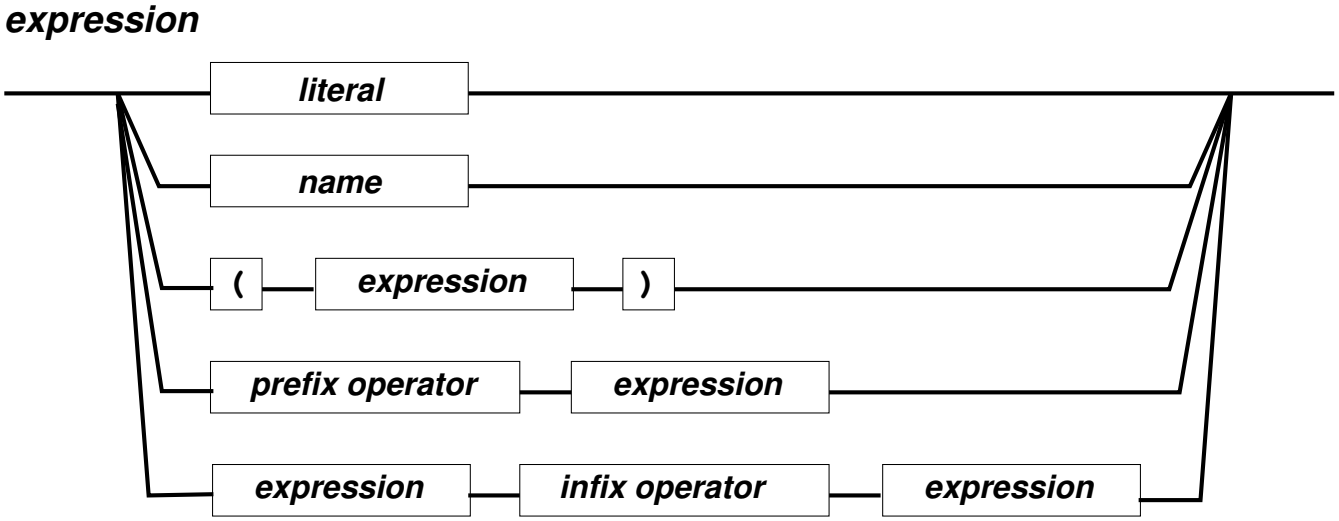
JavaScript Interpreter: Case Study

"*In web browsers, the JavaScript interpreter is particularly prone to security issues; in Mozilla Firefox, for instance, it encompasses the* **majority of vulnerability fixes**...*"*

"*... The problem, however, is that fuzzed input to a JavaScript interpreter must follow the syntactic rules of JavaScript. Otherwise, the JavaScript interpreter will reject the input as invalid, and effectively* **restrict** *the testing to its* **lexical and syntactic analysis**, *never reaching areas like code transformation, in-time compilation, or actual execution.* "

See "Fuzzing with Code Fragments", Holler *et al.*

JavaScript Interpreter: Model-Based Testing



- **Language Grammar** for Javascript

  - Provides **model** for test generation

  - That is, only generate tests **matching grammar**

- **Q)** *What is this model abstracting / focusing on?*

Further Reading

- “Levels of Fuzzing”
  `https://blog.regehr.org/archives/1039`
- “Taming compiler fuzzers”, Chen, Groce, Zhang, Wong, Fern, Eidt, Regehr, PLDI’13
- “Differential Software Testing”, McKeeman, Digital Technical Journal
- “Fuzzing with Code Fragments”, Holler, Herzig, Zeller. USENIX’12
- “JCrasher: an automatic robustness tester for Java”, Csallner and Smaragdakis. *Software Practice and Experience*, 2004.