Equivalence Class Testing, Decision Table Testing
Upcoming Events

- February 28: Visit by Microsoft Studios
  - Game crits by staff from Microsoft Studios
  - Discussion of career opportunities

- YetiZen: Everything you thought you knew about start-up building & fundraising told by Storm8 & Charles Hudson
  - In San Francisco, Thursday, Feb. 23, 6pm
  - http://www.eventbrite.com/event/2932897375
  - See me/email me for free reg. code

- Prom Week released last week
  - Please consider voting for it in the IGF Audience Choice awards: http://igf.com/audience.php
Upcoming deadlines

- **Thursday (February 23)**
  - Sprint 2 ends
  - Game playtesting plan
  - Web site framework (make sure website URL is in your Sprint 2 report)

- **Friday (February 24)**
  - Sprint 2 report due
  - Team status reporting due

- **Sprint 3 begins:: Friday, February 24**

- **Monday (February 27)**
  - Sprint 3 plan due
Game Playtesting Plan

- 1-2 page document, now due Tuesday
- Answer the following questions:
  - Who will conduct tests and write playtest report?
    - Should have 1-2 people on your team for whom this is a major task
  - How will you recruit testers?
    - Just sending out emails isn’t likely to get a lot of responses
    - Need to personally recruit friends, friends of friends, put up flyers, etc.
    - Whole team effort
  - When/Where will you conduct playtests?
    - What day of the week will you conduct playtests? If in the lab, where, exactly?
  - What specific gameplay issues do you want to focus on in the first few weeks of playtesting?
  - Do you have any special equipment needs for conducting playtests?
Lab cleanup schedule

- This week: Hello World
- Next week (and through end of the quarter): Puzzle Defense

**Team duties:**
- Ensure overflowing trash cans are emptied to bin outside in 3rd floor courtyard (anytime during week)
- By 5pm Monday (Tuesday this week, due to holiday) and 5pm Friday (unless things get out of control, then more often):
  - Pick up food containers, bottles, etc.
  - Pick up stray craft materials, pens, etc and return to drawers
  - Clean off tables in conference rooms and big circular table
  - Report any major soda/food spills to me, so we can call cleanup crews
  - Put controllers/game boxes/etc. away (tidy up game area)
  - Report any cleaning materials needed
Next quarter all team artists should take class Art 105 (Special Topics in Drawing)

- The instructor is listed as “Staff” online, but is actually Richard Wohlfeiler <rawohlfe@ucsc.edu>
- Team artists who are not in 172 and who are non-Art majors should also take the class
- The class is closed, and each artist will need a permission code to get in
- I need to send Art Dept. a list of the names of students who will receive permission codes
Ultra top secret info about classes next quarter

- I’ll be sending this out soon, but you heard it here first
- Lots of new classes next quarter
  - CS 26 (Intro. Computer Animation)
    - 3D animation with Blender (Yonge), CS 25 as prereq.
  - CS 119 (Software for Society)
    - Applying computing to social issues (Davis)
  - CS 121 (Mobile Applications) – CGE
    - Android programming (de Alfaro)
  - CS 162 (Advanced Computer Graphics / Animation Lab) - CGE
    - Continuation of CS 160, focused on computer animation (Pang)
  - CS 179 (Game Design Practicum) - CGE
    - Make 3 games using Kinect (Whitehead)
Slots for game critiques on Tuesday

- Microsoft visit is next Tuesday
- Game demos in the game lab
- ~5 min presentation, ~5 min playing game, ~10 min q&a, crit
- 2:00 : Firewall
- 2:20 : Sonar
- 2:40 : Hello World
- 3:00 : MicroVentures
- 3:20 : Puzzle Defense
- 3:40 : Chroma
- 4:00 : Devil’s Bargain
There are two fundamental approaches to testing software:

Functional (Black Box)
- No detailed knowledge of internal implementation
- Treats system as function mapping inputs to outputs

Structural (White Box or Clear Box)
- Can use detailed knowledge of internal implementation detail
- Can “see” inside the implementation
Types of Functional Tests

- Boundary Value Testing
  - Test values just before, on, and after some input boundary
  - E.g., -1, 0, 1

- Equivalence Class Testing
  - Provide representative samples within a large space of similar ("equivalent") inputs

- Decision-Table Testing
  - Create a table that lists all possible outcomes for a set of inputs
  - Use this table to drive test case selection
Equivalence Class Testing

- Motivating idea:
  - Many inputs to a program will exercise the same path through its source code.
  - It is usually the case that, for a given path through the code, one set of inputs is just as good as any other set of inputs for a test case.
    - Not always true – recall that for the triangle problem, for some implementation approaches, input values near Maxint would lead to integer overflow.
  - So, if it is possible to partition the input space (or output space), then the only tests necessary are one per equivalence class.

- “The idea of equivalence class testing is to identify test cases by using one element from each equivalence class”
Example

- Consider collision detection for the hypothetical game, “Lego Gradius”
  - Vic Viper needs to support parts of the ship being knocked off
  - A hit to the side wings by a point bullet only knocks off a wing
  - A hit to the main body destroys the entire ship
  - The tail area and front wings are immune to hits

Two input variables:
bul_x and bul_y (bullet location)

a ≤ bul_x ≤ d
intervals:
[a, b), [b, c), [c, d]

e ≤ bul_y ≤ h
intervals:
[e, f), [f, g], (g, h]
Weak Normal Equivalence Testing

- Pick one value from each equivalent class for each input variable
  - Test case 1: \([a,b) , [e,f)\]
  - Test case 2: \([b,c) , [f,g]\)
  - Test case 3: \([c,d) , (g,h)\]
- Problem: misses some equivalence classes

Two input variables:
bul_x and bul_y (bullet location)

\[a \leq bul_x \leq d\]
intervals:
\([a,b) , [b,c) , [c,d]\)

\[e \leq bul_y \leq h\]
intervals:
\([e,f) , [f,g]\) \((g,h)\]

---

UC SANTA CRUZ
Strong Normal Equivalence Testing

- Pick one value from the combination of each equivalence interval for one variable with every equivalence interval for the second variable
  - Test case 1: \([a,b), [e,f)\]
  - Test case 2: \([a,b), [f,g)\]
  - Test case 3: \([a,b), (g,h)\]
  - Test case 4: \([[b,c), [e,f)]]\) …
- Problem: doesn’t test values outside of intervals…

Two input variables:
bul_x and bul_y (bullet location)

- \(a \leq \text{bul}_x \leq d\)
- Intervals: \([a, b), [b, c), [c, d]\)
- \(e \leq \text{bul}_y \leq h\)
- Intervals: \([e, f), [f, g], (g, h]\)
Weak Robust Equivalence Testing

- Pick one value from each equivalent class for each input variable, and include invalid values as equivalence classes
  - Test case 1: \(< a, [e,f)>\)
  - Test case 2: \([a,b), [e,f)>\)
  - Test case 3: \([b,c), [f,g)]\)
- Problem: misses some equivalence classes

Two input variables:
bul\_x and bul\_y (bullet location)

- \(a \leq \text{bul\_x} \leq d\)
  - Intervals: \([a, b), [b, c), [c, d]\)
- \(e \leq \text{bul\_y} \leq h\)
  - Intervals: \([e, f), [f, g], (g, h]\)

Test case 4: \([c,d], (g,h)]\)
Test case 5: \(> d, > h\}
Test case 6: \([b,c), < e\)
Strong Robust Equivalence Testing

- Pick one value from the combination of each equivalence interval for one variable with every equivalence interval for the second variable, including invalid values as equivalence classes
  - Test case 1: \([a,b), [e,f]\)  
  - Test case 2: \([a,b), [f,g]\)  
  - Test case 3: \([a,b), (g,h]\)  
  - Test case 4: \([b,c), [e,f]\] …
- Problem: none, but results in lots of test cases (expensive)

Two input variables:
bul_x and bul_y (bullet location)

- \(a \leq \text{bul}_x \leq d\)
- Intervals: \([a, b), [b, c), [c, d]\)  
  - Also: \(< a \) and \( > d\)

- \(e \leq \text{bul}_y \leq h\)
- Intervals: \([e, f), [f, g], (g, h]\)  
  - Also: \(< e \) and \( > h\)
Triangle problem (revisited)

- Triangle problem
  - A program reads three integer values from the console. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral.

- Recall:
  - Equilateral triangle:
    - Three equal sides, three equal angles (all 60 degrees)
  - Isosceles triangle:
    - Two equal sides, two equal angles
  - Scalene triangle:
    - No equal sides, no equal angles

- Use Output classes
  - In this case, is best to develop test cases based on output equivalence classes
  - What are they?
  - What are examples of weak normal, strong normal, weak robust, and strong robust test cases (assume max side length of 200)
Decision Table Testing

- Core idea:
  - Use a table format to capture the relationship between input conditions and output actions
  - Then, use combination of conditions and actions to develop test cases
  - Benefit: tables provide a rigorous way of specifying conditions and actions, since omissions are very clear (an empty part of the table)

- Decision table format:

<table>
<thead>
<tr>
<th>Stub portion</th>
<th>Entry portion</th>
</tr>
</thead>
</table>
Decision Table Format

- **Conditions** describe logical states the software can be in
  - Often describe ranges of inputs
- **Actions** describe possible things the software can do
  - Often are the outputs
- **Rules** describe what actions (if any) occur for a given set of conditions

<table>
<thead>
<tr>
<th>Conditions (condition stub)</th>
<th>Rule 1</th>
<th>Rule 2</th>
<th>...</th>
<th>Rule n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actions (action stub)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Triangle Problem as Decision Table

<table>
<thead>
<tr>
<th>C1: a,b,c form a triangle?</th>
<th>F</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2: a=b?</td>
<td>--</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C3: a=c?</td>
<td>--</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C4: b=c?</td>
<td>--</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>A1: Not a triangle</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2: Scalene</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3: Isosceles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4: Equilateral</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5: Impossible</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Triangle Problem as Decision Table

<table>
<thead>
<tr>
<th>C1: a, b, c form a triangle?</th>
<th>F</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2: a = b?</td>
<td>--</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C3: a = c?</td>
<td>--</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C4: b = c?</td>
<td>--</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>A1: Not a triangle</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2: Scalene</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A3: Isosceles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>A4: Equilateral</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5: Impossible</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generating Test Cases from Decision Table

- For each row in the table, pick actual values that satisfy the conditions, and then verify the output state(s) match the actions.

- In-class:
  - Find example test cases from the complete decision table for the triangle problem
  - What are the pros and cons of these test cases?
Random Testing

- One final black box testing technique is to use a random number generator to create inputs.
- Requires a stopping criteria
  - Could generate random numbers forever – how many do you want?
- Pro:
  - Can make a lot of tests, cheaply
  - Explore larger, variable part of input space
- Con:
  - Need to control seed to ensure test cases are repeatable (otherwise vary with each run)
  - Tests are more computationally expensive to run
  - Tests may require more work to create for the first time

- Example: random testing the triangle problem
  - Randomly pick triangle side values
  - Stopping criteria: continue until every possible output has been seen at least once
  - Example (from book):
    - 1289 test cases, of which 663 are nontriangles, 593 are scalene, 32 are isosceles, and 1 is equilateral
Oracles

- No, not the database company
- Think Oracle of Delphi
  - Tells the truth about your future

- A **test oracle** is a function that determines whether the program’s output for a given set of inputs is correct or incorrect.

- A human being often acts as the test oracle.
  - Either manually testing code, or determining what the outputs should be for an automatically run regression test with static inputs
- For random testing, the test oracle must be a computational function.
  - It can often be as difficult to make the test oracle as it is to create the test