Software Testing, Quality Assurance and Maintenance Winter 2010

Lecture 14 — February 3, 2010

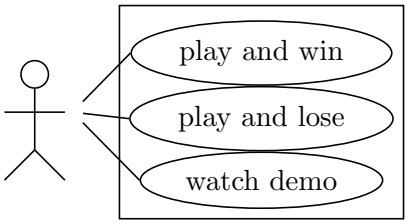
Patrick Lam *version 1*

Graph Coverage for Use Cases

Key Idea. Create graphs from use cases and cover them.

We will actually use activity diagrams, not use cases (so that we have graphs with more than 1 path.) Note that UML Class Diagrams aren't so helpful.

Example. Outrun, a videogame from 1986. Set of use cases (see below) isn't very useful for testing, although we could use node coverage.



Videogame player actor and use cases.

Here is a more elaborated description of a use case.

Use Case Name:	Play and Win
Summary:	Player successfully drives to the goal in allotted time.
Actor:	Player
Precondition:	Game is in demo mode.

Description.

1. Player inserts coins into system.
2. Player hits start button.
3. If sufficient credit, deduct credit and proceed to difficulty and music selection.
4. Player selects difficulty and music.
5. System loads appropriate data and enters main game loop.
6. System starts timer and accepts game input.
7. Player steers car through traffic to destination on time.

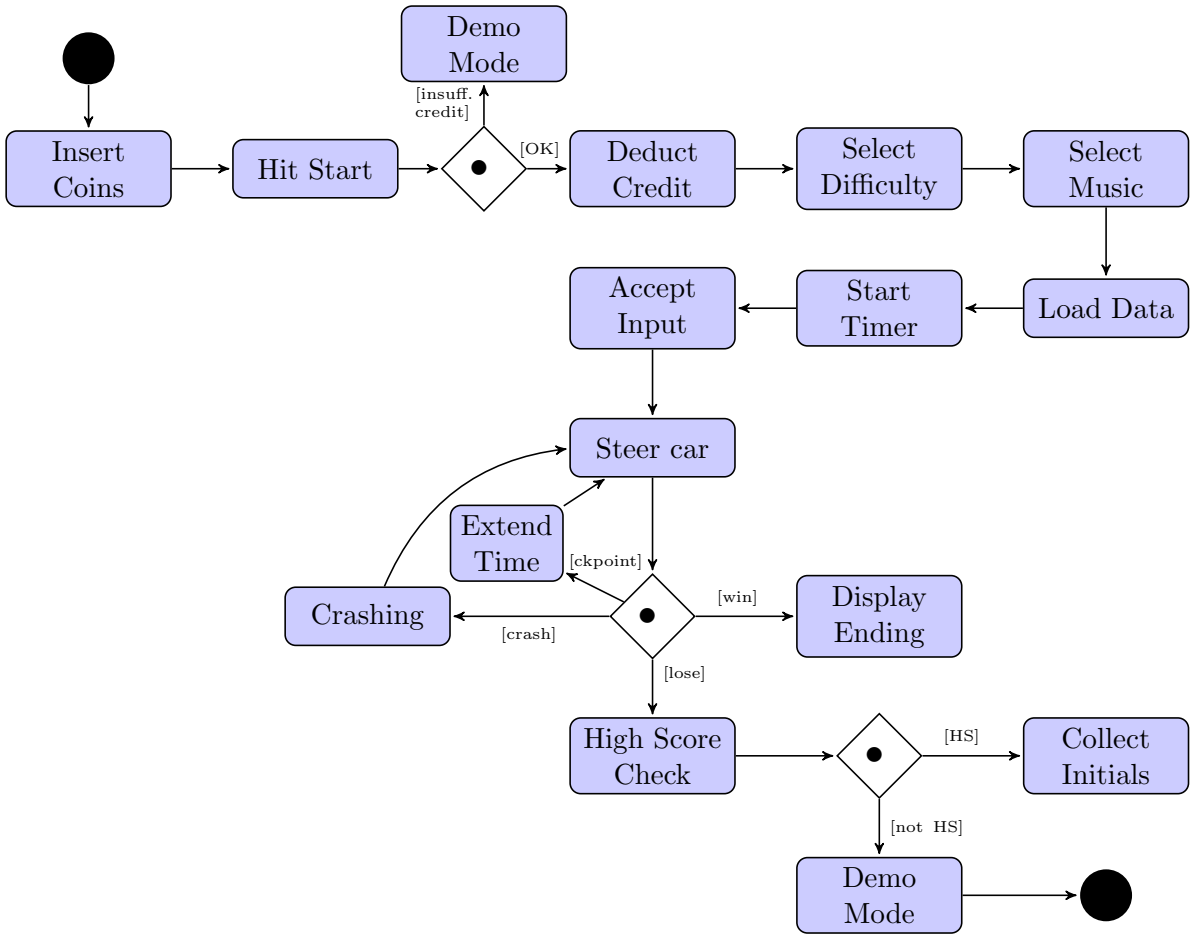
8. When player passes a checkpoint, system extends timer.
9. System displays appropriate winning ending.
10. If player has a high score, get initials and add to highscore list.
11. Return to demo mode.

Alternatives. Some other possibilities might be:

Postcondition. Game system in demo mode.

Activity diagram

Here is an activity diagram for this use case.



Summary Discussion of Graph Coverage

You should be able to do the following with graphs:

Strengths and weaknesses of Graph Coverage:

Summarizing Structural Coverage:

- generic; hence broadly applicable
- uses no domain knowledge

Summarizing Dataflow Coverage:

- definitions and uses are a concept that often reappears.

Miscellaneous other notes:

- Control-flow graphs are tractable for single methods, but not generally more than that. Use call graphs to represent multiple methods, hiding details of each method.
- When we want to test du-paths across multiple design elements, use first-use/last-def heuristic.
- Testing based on specifications: sequencing constraints can be hard to find, and FSMs may be unwieldy.