Kodu Module 5: State Machines

David S. Touretzky
Version of 19 June 2015

<u>Learning Goals</u>

- Programs can comprise multiple pages.
- State machine diagrams are a helpful way to represent such complex programs.
- The states, which correspond to pages, are drawn as circles.
- The transitions, which correspond to "switch to page" actions, are drawn as labeled arrows.

<u>Time Required:</u> 2 hours

<u>Worlds</u>

- State1
- Apple3

<u>Manipulatives</u>

- For the CS Unplugged activity:
    - To represent apples: two red balls and two blue balls. You can also use red and blue balloons.
    - A sign board or piece of poster board where you can write on both sides.
- No tiles

<u>Flash Cards</u>

- Show Page As Color

Part 1: Trying To Reach The Castle – and Failing

1. Load and run the State1 world. Explore it. What is swimming in the moat around the castle?
2. We want to get the kodu to the castle. This is harder than it looks. Try this program:

```
[1] WHEN see (objects) castle DO move toward
```

3. What happens when the kodu tries to go directly to the castle? (It falls in the moat and gets eaten.)
4. To keep the kodu out of the moat, let's try sending it to the red tree first. From there it should be able to safely travel to the castle. Try this program:

```
[1] WHEN see red (objects) (more) tree DO move toward
[2] WHEN see castle DO move toward
```

5. What happens when you run this program? (The kodu gets stuck at the red tree; the second rule always loses out to the first rule.)
6. *(Optional: skip this for younger students.)* What we want is for the kodu to reach the red tree and then forget about that goal and focus instead on getting to the castle. Here is an attempt to do that, but it won't work. Notice that the rule to move toward the red tree now comes *after* the rule to move toward the castle, so the castle can take priority when the kodu bumps the tree.

```
[1] WHEN bump red tree DO
    ↳ [2] WHEN see castle DO move toward
[3] WHEN see red tree DO move toward
```

7. What happens when you run this program?
8. Here's why the kodu still gets stuck at the tree: As soon as it bumps the tree rule 2 can fire. But when the kodu takes one step toward the castle, "bump red tree" in rule 1 is no longer true, so rule 2 can't continue to fire because it depends on rule 1. So rule 3 runs again and takes the kodu back to the tree, and the process repeats, over and over. We must find another solution.

Part 2: Programs Can Have More Than One Page

1. In the rule editor, look at the top of the screen and you'll see the number "1". This is the page number. Up to now, all our programs have been written on page 1, but Kodu programs can have multiple pages. The kodu can only be on one page at a time, and it only pays attention to the rules on that page, but it can move from one page to another using the "switch to page" action.
2. In the rule editor, press the right shoulder button to move to the next page. The left shoulder button moves to the previous page. How many pages can a program have? (Answer: 12.)
3. Another way to move between pages is to move the pencil (using the left stick) up until it reaches the page number. Then you can move the left stick left or right to move between pages.

Part 3: Adding A Second Page To The State1 Program Solves the Problem

1. In the rule editor, go back to page 1 and enter the following program:

   PAGE 1:

   [1] WHEN see red tree DO move toward

   [2] WHEN bump red tree DO switch to page 2

2. Move to page 2 and enter the following:

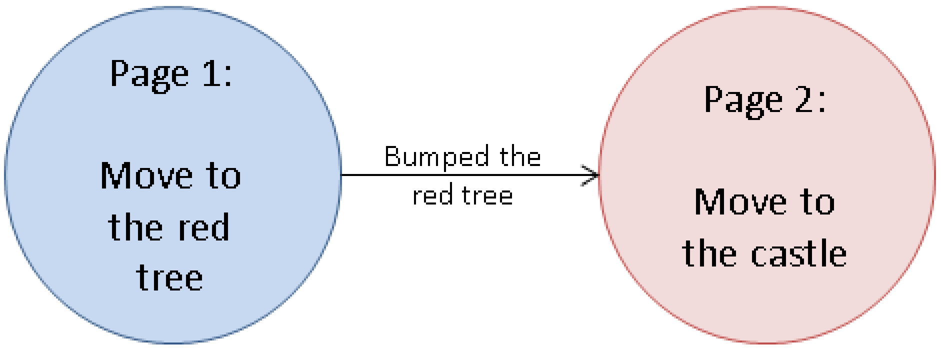
   PAGE 2:

   [1] WHEN see castle DO move toward

3. Now when you run the program, the kodu should reach the castle.

Part 4: State Machines

1. On page 1 the kodu's goal is to reach the red tree. When it does so, it switches to page 2, which has a different set of rules. On page 2 the kodu's goal is to reach the castle; it no longer cares about the red tree.
2. We can describe the kodu's change in behavior using a *state machine diagram*. Each page of the program constitutes a state, drawn as a circle. Each "switch to page" rule is drawn as an arrow from one state to another. The arrow is labeled with the condition controlling when the "switch to page" action will be taken. Here is the diagram for the program we just wrote:



3. When you start the program, the kodu always begins on page 1, which is why we say that state 1 is the *start state*.
4. How many states can the kodu have? (The kodu can have up to 12 states because there are 12 pages.)

Part 5: The "Show Page As Color" Idiom

1. We can have the kodu tell us its state by changing its color to indicate the page it's currently on.
2. Go into the rule editor and compare the page numbers for page 1 and page 2. Page 1 is light blue, while page 2 is pink.
3. Examine the "Show Page As Color" flash card. If we want the kodu to tell us its state, it should color itself blue whenever it's on page 1, and pink when it's on page 2.
4. Modify your program to add the Show Page As Color idiom:

```
PAGE 1:
  [1] WHEN DO (actions) color me blue
  [2] WHEN see red tree DO move toward
  [3] WHEN bump red tree DO switch to page 2

PAGE 2:
  [1] WHEN DO color me pink
  [2] WHEN see castle DO move toward
```

5. Run the program and watch the kodu to see it change from state 1 (blue) to state 2 (pink).

Part 6: A More Challenging Path to the Castle

Distribute the State1 World handout and have the students solve the problem on their own.

Part 7: States in Pac-Man (Optional)

1. Tell students that switching states is a key part of how computer games are programmed. Ask them if they can think of an example from the game Pac-Man.
2. Run the Google Pac-Man doodle from this page: **http://goo.gl/NfX6J**. Press "Insert Coin" to play.
3. Use the arrow keys to steer the Pac-Man to eat one of four the energizer dots.
4. Notice that all the ghosts turn blue and instead of chasing the Pac-Man, they run away from him. The ghosts have **changed state**. Instead of their normal Pursue and Consume strategy, they are fleeing!
5. Fun trivia: Pac-Man was released in 1980. The ghosts's names are Blinky, Pinky, Inky, and Clyde.

Part 8: Drawing A State Machine From Rules

Distribute the Drawing A State Machine From Rules handout and have the students solve the problem on their own.

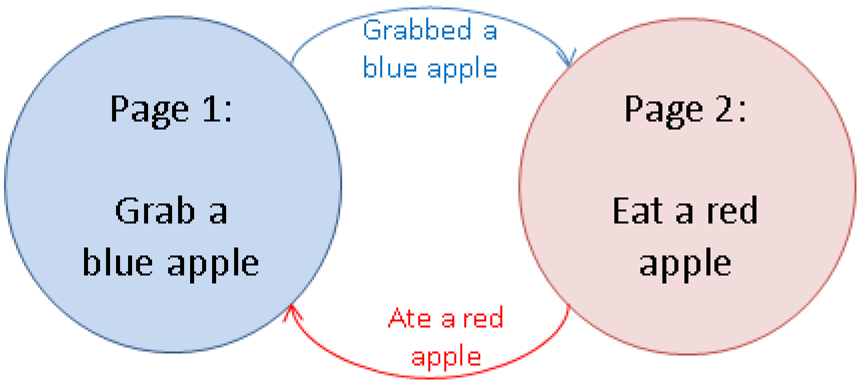
Answer to the puzzle: the problem is that there is no way to reach page 3.

Part 9: A State Machine With a Cycle

1. Load the Apple3 world. You've seen similar worlds before: the kodu has to eat the red apples and grab the blue ones. But now we introduce a new twist: it has to alternate between the two colors. First it must grab a blue apple, then eat a red one, then grab another blue one, then eat another red one, and so on. If it ever grabs two blue apples or eats two red apples in a row, it dies.
2. We could try to eat all the apples with just one page of rules, but it won't work:

```
[1] WHEN see blue apple DO move toward
[2] WHEN bump blue apple DO grab it
[3] WHEN see red apple DO move toward
[4] WHEN bump red apple DO eat it
```

3. What happens when you run this program? (The kodu tries to grab all the blue apples first, because rule 1 takes priority over rule 3.)
4. What happens if you switch the order of rules 1 and 3? (The kodu tries to eat all the red apples first.)
5. To solve this problem we need a state machine with two states:



6. Here are the rules for the first part of the program:

```
PAGE 1:
 [1] WHEN DO color me blue
 [2] WHEN see blue apple DO move toward
 [3] WHEN bump blue apple DO grab it
     ↳ [4] WHEN DO switch to page 2
```

7. Run this program and see what happens.
8. Why does the kodu stop after it grabs the blue apple? (Because it's on page 2, which has no rules.)
9. Complete the program by writing the rules for page 2. Use the Show Page As Color idiom.
10. Run your program and see what happens when all the apples are gone.

Part 10: CS Unplugged Simulation of Apple3

"CS Unplugged" is a technique for teaching computer science concepts using household items as physical manipulatives. The creators, Tim Bell, Ian Whitten, and Michael Fellows, call it "computer science without a computer." See the http://csunplugged.org web site for a free book of CS Unplugged activities.

The goal for this part of the lesson is to have students simulate a character maintaining a state and changing its state based on what its rules tell it to do.

Distribute the Apple3 Simulation handout and have students perform the activity.

Part 11: Review and Assessment

Have students complete the questionnaire for this lesson.