

CS51 Project 2a: Don't Drop the Ball

Due: Tuesday, March 7th 2009 at 11:59 PM

Total Points: 45 (including 10 style points)

This is the first part of your second long-term project for the semester. These projects are intended to get you thinking about issues of large-scale design and iteration of code through multiple versions and stages, as well as what complications and benefits arise as a result of working with a partner.

1 Project Overview

This project will focus on building parts of a simulation engine and artificial intelligence agents within it that interact and compete with one another. In this particular assignment you will subclass and override methods of an existing agent interface. Your agent will use Dijkstra's algorithm to calculate the shortest path to its goal in order to catch a "ball" agent before it lands.

2 Introduction to the World

We have provided a class containing the game rules for this simulation (called `ball-catch%`), and a simulation engine (`engine-2d%`¹). To see it run, we just need to create a graph, create a new engine, and send the engine the "start" command.

```
;; make a graph  
(define graph (graph-add-edge 'a 'b (graph-add-edge 'b 'a graph-new)))  
(define engine (make-object engine-2d% graph ball-catch%))  
(send engine init-engine)
```

Alone, this isn't too interesting. We can also add some agents to the graph. We have given you two agents already.

¹Hopefully with a 3d version coming soon!

```
(define ball (make-object ball-agent%))
(define a1 (make-object graph-agent%))
(send engine add-agent-at-node ball 'a)
(send engine add-agent-at-node a1 'a)
(send engine draw)
```

With the above code you will see agents appear on the graph. If you step the simulation forwards, you will see that the ball agent will fly from off the graph onto a node. If no other agent is there, it will sit there until it is caught. The graph-agent doesn't move. Your goal will be to construct an agent that will be able to “catch” the ball each time before it lands at a particular node.

If you run this code several times, you may also note that the graph layout changes each time. The engine is dynamically creating a new graph layout each time it is run. When you begin running your code on more complicated graphs, keep in mind that the layout may not always look pretty, and that the layout for more complicated graphs will take longer to create. For this assignment, assume that your graphs are fully connected (i.e., there are no unreachable or disconnected nodes).

3 Agents

Agents in this simulation all inherit from the class `agent-class%`.

Note, `new-goal` will only be called by the engine when the agent is currently on a node in the graph (i.e., not in transition between nodes).

```
(define agent-class%
  ...

  ;; Sets the exterior and interior colors of the agent, respectively.
  (define/public (get-default-pen) (make-object pen% "GREEN" 2 'solid))
  (define/public (get-default-brush) (make-object brush% "GREEN" 'solid))

  ;; Draws the bitmap representing the agent when it is first initialized.
  ;; bm-dc represents the drawing context for a 10x10 bitmap.
  (define/public (init-bitmap bm-dc) ...)

  ;; Can be used to modify the bitmap everytime draw-bitmap is called, for
  ;; instance, for animation. Currently does nothing.
  (define/public (draw-bitmap bm-dc) (void))

  (define/public (get-type) 'agent-class)

  ;; Returns a new node in the graph to move to.
  (define/public (new-goal query-obj) (error "Not implemented"))
)
```

We have also defined a class `graph-agent%`, which is very similar to `agent-class%` with the only exception that, when picking a new goal, it forces its subclasses to pick only goals that are immediately reachable from the node the agent is currently at.

```
(define graph-agent%
  ...

  ;; Method to override. Picks a new node to go to – only called when
  ;; the agent is currently at a node on the graph.
  (define/public (new-goal-node query-obj)
    (error "YOUR-IMPLEMENTATION"))

  ;; Calls new-goal-node, and requires that the goal node is a neighbor
  ;; of the current node. If not, returns an error.
  (define/override-final (new-goal query-obj) ...))

(define/override (get-type) 'graph-agent)
)
```

Everything looking good so far? The final bit of information we need to give you for this assignment, is for you to understand what this argument `query-obj` is doing in the `new-goal` methods above. The `query-obj` is an instance of the game-rules initialized with an agent, and you can use this to get simulation-specific information about the state of the world by using: (`send query-obj function`). You will need this object to get the graph that is in the engine, the position of the current agent, and the node in the graph that the ball is heading towards.

```
(define ball-catch%
  ...

  ;; Returns the graph in the engine, and the node in the graph that the
  ;; agent is currently on.
  (define/public (get-graph) ...)
  (define/public (get-node) ...)

  ;; Returns what node in the graph the ball-agent is currently moving
  ;; towards. Returns #f if there is no such agent or goal in the
  ;; engine.
  (define/public (get-ball-goal) ...) )
```

We suggest reviewing the lecture notes from Greg's lecture on classes if the above seems confusing or unfamiliar.

4 Writing Your Agents

(34 points total)

We have given you your own agent class to fill in, `my-agent%`.

Exercise 1.

34 points

(a) [2 points]

Customize your agent! Override the `get-default-pen` and `get-default-brush` methods in your agent class to change the exterior and interior colors of your agent, respectively. If you are feeling particularly creative, you may also override the `init-bitmap` or `draw-bitmap` methods instead – remember though, you only have 10 x 10 pixels to work with. Look up the classes `pen%`, `brush%`, and `dc%` for information about pens, brushes, and what drawing commands are available. (A fast way to access the documentation is to type in any of these names in Dr. Scheme and then press F1).

(b) [4 points]

Override `new-goal-node` so that your agent does a random-walk along the graph. In other words, when `new-goal-node` is called, your agent should pick a random legal node to move to from its current node.

(c) [20 points]

Implement Dijkstra's algorithm, as discussed in lecture and section. We are leaving the design of your algorithm, as well as the structure of what is being returned, somewhat up to you. Take a step back and plan out how you're going to approach the problem before you actually begin coding. What helper functions are you going to use? We've started you off below: you'll want a helper function to update the costs of each neighbor of a node. The last argument(s) you'll need will depend on what data structures you're using; feel free to use whatever structures you want. You may define (a) new struct(s) if you would like to.

You may make use of both the dictionary and set modules provided. You will need to be able to use your function to determine not only the shortest distance between the source node and any other node in the graph, but also a path that starts at the source node and realizes that lowest cost to each destination node. Thus, in addition to your Dijkstra's function, you should end up with a function that takes two nodes and returns the best path to take to get from one to the other.

When you're done, be sure to copy your implementation into your agent's class definition at the end of your `proj2a` scheme file.

(d) [8 points]

Override `new-goal-node` so that your agent attempts to "catch" the ball before it lands. You will need to comment out or rename (but don't delete!) your random walk code. Don't forget that the `get-ball-goal` method of `query-obj` returns false if no ball or ball-goal exists - you will need to handle this as well.

To test your work, you might want to create an engine and add a ball and your agent to it. Run it through several time steps – if you've programmed it correctly, you should see your agent "catch" the ball-agent the majority of the time before the ball lands. Sample output from running our solution for about 40 timesteps is below:

