C.4 Control Systems Solution

C.4.1 In-lab section

1. The following m-file does the job:

   ```
   function selection = select(array)

   % SELECT - Given a cell array, randomly select one
   % element and return it.

   % Generate a random index.
   index = floor(1 + rand * length(array));

   % There is a small chance that the index will be too
   % large (if rand happens to return 1.0), so we have to
   % check.
   if (index == (length(array) + 1)) index = length(array); end

   selection = array(index);
   ```

   This can be used as follows:

   ```
   >> for i=1:10 x(i) = select(letters); end
   >> x

   x =

       'a'    'a'    'd'    'c'    'e'    'c'    'c'    'e'    'c'    'b'
   ```

   The following m-file does the job:

2.
   ```
   function r = chooserow(a)
   % COMPOSE - return a randomly chosen row of the
   % argument cell array.

   % Get the size of the argument.
   [n, m] = size(a);

   % Generate a random index.
   index = floor(1 + rand * n);

   % There is a small chance that the index will be too
   % large (if rand happens to return 1.0), so we have to
   % check.
   ```

```
if (index == (n + 1)) index = n; end

r = a(index,:);
```

Here we apply it several times:

```
chooserow(t)

ans =

    'upper left'    'upper right'

 chooserow(t)

ans =

    'lower left'    'lower right'

 chooserow(t)

ans =

    'upper left'    'upper right'
```

3. Here is a modified *update* function that returns a cell array:

```
% PETUPDATES - A function representing the state update of a vir-
tual pet.
% The first argument must be in {'absent', 'pet', 'feed', 'time passes'}
% The second argument must be in {'happy', 'hungry', 'dies'}
% The returned value is a two-element cell array, where the first
% element next state of the pet, and the second element
% is the output of the state machine.

function r = pet(state, in)

% The default behavior is to stutter.
r = {state, 'absent'};

switch(state)
case 'happy'
    switch(in)
    case 'pet'
        r = {state, 'purrs'};
    case 'feed'
```

```
            r = {state, ’throws up’};
        case ’time passes’
            r = {’hungry’, ’rubs’};
        end
    case ’hungry’
        switch(in)
        case ’feed’
            r = {’happy’, ’purrs’};
        case ’pet’
            r = {state, ’bites’};
        case ’time passes’
            r = {’dies’, ’dies’};
        end
    case ’dies’
        r = {state, ’dies’};
    end
```

Here is a modified program to run the pet (without the driver):

```
% RUNPET - Execute the virtual pet state machine

% initial state.
petstate=’happy’;

% loop forever.
while 1
    % Get the user input as a string.
    str=input(’enter one of absent, pet, feed, time passes: ’,’s’);

    % If the user entered quit or exit, then break the loop.
    if strcmp(str,’quit’) break; end
    if strcmp(str,’exit’) break; end

    % Update the pet.
    r = petUpdates(petstate, str);
    petstate = r{1};

    % print what the pet does.
    disp(r{2})
    if strcmp(petstate, ’dies’) break; end
end
```

4. Only one line changes:

```
% PETUPDATES - A function representing the state update of a vir-
tual pet.
```

```
% The first argument must be in {’absent’, ’pet’, ’feed’, ’time passes’}
% The second argument must be in {’happy’, ’hungry’, ’dies’}
% The returned value is a two-element cell array, where the first
% element next state of the pet, and the second element
% is the output of the state machine.

function r = pet(state, in)

% The default behavior is to stutter.
r = {state, ’absent’};

switch(state)
case ’happy’
    switch(in)
    case ’pet’
        r = {state, ’purrs’};
    case ’feed’
        r = {state, ’throws up’};
    case ’time passes’
        r = {’hungry’, ’rubs’};
    end
case ’hungry’
    switch(in)
    case ’feed’
        r = {’happy’, ’purrs’; ’hungry’, ’rubs’};
    case ’pet’
        r = {state, ’bites’};
    case ’time passes’
        r = {’dies’, ’dies’};
    end
case ’dies’
    r = {state, ’dies’};
end
```

In the program that runs the cat, again, only one line changes:

```
% RUNPET - Execute the virtual pet state machine

% initial state.
petstate=’happy’;

% loop forever.
while 1
    % Get the user input as a string.
    str=input(’enter one of absent, pet, feed, time passes: ’,’s’);
    % If the user entered quit or exit, then break the loop.
    if strcmp(str,'quit') break; end
    if strcmp(str,'exit') break; end

    % Update the pet.
    r = chooserow(petUpdates(petstate, str));
    petstate = r{1};

    % print what the pet does.
    disp(r{2})
    if strcmp(petstate, 'dies') break; end
end
```

Here is a sample run:

```
>> runpet
enter one of absent, pet, feed, time passes: time passes
rubs
enter one of absent, pet, feed, time passes: feed
purrs
enter one of absent, pet, feed, time passes: time passes
rubs
enter one of absent, pet, feed, time passes: feed
rubs
enter one of absent, pet, feed, time passes: feed
rubs
enter one of absent, pet, feed, time passes: feed
rubs
enter one of absent, pet, feed, time passes: feed
rubs
enter one of absent, pet, feed, time passes: feed
purrs
```

5. Here is a modification of the cascade composition that uses the nondeterminstic cat instead of the deterministic one:

```
% DRIVEPET - Execute the virtual pet state machine composed
% in cascade with the driver state machine.

% initial state.
driverstate='happy';
petstate='happy';

% loop 10 times, since this is automatically driven.
```

```
for i=1:10,
    % Update the state of the driver and get its output.
    % The input to the driver is always ’1’.
    [driverstate, petinput] = driver(driverstate, ’1’);

    % update the state of the pet and get its output.
    r = chooserow(petUpdates(petstate, petinput));
    petstate = r{1};

    % Display the output of the pet.
    disp(r{2})
end
```

Here is a typical run:

```
>> drivepet
rubs
purrs
rubs
purrs
rubs
purrs
rubs
rubs
dies
dies
```

Inevitably, we get two ’rubs’ in a row and the cat dies.