```
% The simpleGameEngine class inherets from the handle class because we
% want the game objects to be updated by their methods, specifically
% my_figure and my_image
classdef simpleGameEngine < handle
    properties
        sprites = {}; % color data of the sprites
        sprites_transparency = {}; % transparency data of the sprites
        sprite_width = 0;
        sprite_height = 0;
        background_color = [0, 0, 0];
        zoom = 1;
        my_figure; % figure identifier
        my_image;  % image data
    end

    methods
        function obj = simpleGameEngine(sprites_fname, sprite_width,
 sprite_height, zoom, background_color)
            % simpleGameEngine
            % Input:
            %  1. File name of sprite sheet as a character array
            %  2. Width of the sprites in pixels
            %  3. Height of the sprites in pixels
            %  4. (Optional) Zoom factor to multiply image by in final
 figure (Default: 1)
            %  5. (Optional) Background color in RGB format as a 3
 element vector (Default: [0,0,0] i.e. black)
            % Output: an SGE scene variable
            % Note: In RGB format, colors are specified as a mixture
 of red, green, and blue on a scale of 0 to 255. [0,0,0] is black,
 [255,255,255] is white, [255,0,0] is red, etc.
            % Example:
            %      my_scene =
 simpleGameEngine('tictactoe.png',16,16,5,[0,150,0]);

            % load the input data into the object
            obj.sprite_width = sprite_width;
            obj.sprite_height = sprite_height;
            if nargin > 4
                obj.background_color = background_color;
            end
            if nargin > 3
                obj.zoom = zoom;
            end

            % read the sprites image data and transparency
            [sprites_image, ~, transparency] = imread(sprites_fname);

            % determine how many sprites there are based on the sprite
 size
            % and image size
            sprites_size = size(sprites_image);
```

```
            sprite_row_max = (sprites_size(1)+1)/(sprite_height+1);
            sprite_col_max = (sprites_size(2)+1)/(sprite_width+1);

            % Make a transparency layer if there is none (this happens
when
            % there are no transparent pixels in the file).
            if isempty(transparency)
                transparency = 255*ones(sprites_size,'uint8');
            else
                % If there is a transparency layer, use remap() to
                % replicate is to all three color channels
                transparency = repmat(transparency,1,1,3);
            end

            % loop over the image and load the individual sprite data
into
            % the object
            for r=1:sprite_row_max
                for c=1:sprite_col_max
                    r_min = sprite_height*(r-1)+r;
                    r_max = sprite_height*r+r-1;
                    c_min = sprite_width*(c-1)+c;
                    c_max = sprite_width*c+c-1;
                    obj.sprites{end+1} =
sprites_image(r_min:r_max,c_min:c_max,:);
                    obj.sprites_transparency{end+1} =
transparency(r_min:r_max,c_min:c_max,:);
                end
            end
        end

        function drawScene(obj, background_sprites,
foreground_sprites)
            % draw_scene
            % Input:
            %  1. an SGE scene, which gains focus
            %  2. A matrix of sprite IDs, the arrangement of the
sprites in the figure will be the same as in this matrix
            %  3. (Optional) A second matrix of sprite IDs of the same
size as the first. These sprites will be layered on top of the first
set.
            % Output: None
            % Example: The following will create a figure with 3 rows
and 3 columns of sprites
            %      drawScene(my_scene, [4,5,6;7,8,9;10,11,12],
[1,1,1;1,2,1;1,1,1]);

            scene_size = size(background_sprites);

            % Error checking: make sure the bg and fg are the same
size
            if nargin > 2
                if ~isequal(scene_size, size(foreground_sprites))
```

```
                    error('Background and foreground matrices of scene
must be the same size.')
                end
            end

            num_rows = scene_size(1);
            num_cols = scene_size(2);

            % initialize the scene_data array to the correct size and
type
            scene_data = zeros(obj.sprite_height*num_rows,
obj.sprite_width*num_cols, 3, 'uint8');

            % loop over the rows and colums of the tiles in the scene
to
            % draw the sprites in the correct locations
            for tile_row=1:num_rows
                for tile_col=1:num_cols

                    % Save the id of the current sprite(s) to make
things
                    % easier to read later
                    bg_sprite_id =
background_sprites(tile_row,tile_col);
                    if nargin > 2
                        fg_sprite_id =
foreground_sprites(tile_row,tile_col);
                    end

                    % Build the tile layer by layer, starting with the
                    % background color
                    tile_data =
zeros(obj.sprite_height,obj.sprite_height,3,'uint8');
                    for rgb_idx = 1:3
                        tile_data(:,:,rgb_idx) =
obj.background_color(rgb_idx);
                    end

                    % Layer on the first sprite. Note that the
tranparency
                    % data also ranges from 0 (transparent) to 255
                    % (visible)
                    tile_data = obj.sprites{bg_sprite_id} .*
(obj.sprites_transparency{bg_sprite_id}/255) + ...
                        tile_data .* ((255-
obj.sprites_transparency{bg_sprite_id})/255);

                    % If needed, layer on the second sprite
                    if nargin > 2
                        tile_data = obj.sprites{fg_sprite_id} .*
(obj.sprites_transparency{fg_sprite_id}/255) + ...
                            tile_data .* ((255-
obj.sprites_transparency{fg_sprite_id})/255);
                    end
```

```
                    % Calculate the pixel location of the top-left
 corner
                    % of the tile
                    rmin = obj.sprite_height*(tile_row-1);
                    cmin = obj.sprite_width*(tile_col-1);

                    % Write the tile to the scene_data array
                    scene_data(rmin+1:rmin+obj.sprite_height,cmin
+1:cmin+obj.sprite_width,:)=tile_data;
                end
            end

            % handle zooming
            big_scene_data = imresize(scene_data,obj.zoom,'nearest');

            % This part is a bit tricky, but avoids some latency, the
 idea
            % is that we only want to completely create a new figure
 if we
            % absolutely have to: the first time the figure is
 created,
            % when the old figure has been closed, or if the scene is
            % resized. Otherwise, we just update the image data in the
            % current image, which is much faster.
            if isempty(obj.my_figure) || ~isvalid(obj.my_figure)
                obj.my_figure = figure();
                obj.my_image =
 imshow(big_scene_data,'InitialMagnification', 100);
            elseif isempty(obj.my_image)  ||
 ~isprop(obj.my_image, 'CData') || ~isequal(size(big_scene_data),
 size(obj.my_image.CData))
                figure(obj.my_figure);
                obj.my_image =
 imshow(big_scene_data,'InitialMagnification', 100);
            else
                obj.my_image.CData = big_scene_data;
            end
        end

        function key = getKeyboardInput(obj)
            % getKeyboardInput
            % Input: an SGE scene, which gains focus
            % Output: next key pressed while scene has focus
            % Note: the operation of the program pauses while it waits
 for input
            % Example:
            %      k = getKeyboardInput(my_scene);


            % Bring this scene to focus
            figure(obj.my_figure);
```

```
            % Pause the program until the user hits a key on the
keyboard,
            % then return the key pressed. The loop is required so
that
            % we don't exit on a mouse click instead.
            keydown = 0;
            while ~keydown
                keydown = waitforbuttonpress;
            end
            key = get(obj.my_figure,'CurrentKey');
        end

        function [row,col,button] = getMouseInput(obj)
            % getMouseInput
            % Input: an SGE scene, which gains focus
            % Output:
            %  1. The row of the tile clicked by the user
            %  2. The column of the tile clicked by the user
            %  3. (Optional) the button of the mouse used to click
(1,2, or 3 for left, middle, and right, respectively)
            %
            % Notes: A set of “crosshairs” appear in the scene’s
figure,
            % and the program will pause until the user clicks on the
            % figure. It is possible to click outside the area of the
            % scene, in which case, the closest row and/or column is
            % returned.
            %
            % Example:
            %      [row,col,button] = getMouseInput (my_scene);

            % Bring this scene to focus
            figure(obj.my_figure);

            % Get the user mouse input
            [X,Y,button] = ginput(1);

            % Convert this into the tile row/column
            row = ceil(Y/obj.sprite_height/obj.zoom);
            col = ceil(X/obj.sprite_width/obj.zoom);

            % Calculate the maximum possible row and column from the
            % dimensions of the current scene
            sceneSize = size(obj.my_image.CData);
            max_row = sceneSize(1)/obj.sprite_height/obj.zoom;
            max_col = sceneSize(2)/obj.sprite_width/obj.zoom;

            % If the user clicked outside the scene, return instead
the
            % closest row and/or column
            if row < 1
                row = 1;
            elseif row > max_row
                row = max_row;
```

```
            end
            if col < 1
                col = 1;
            elseif col > max_col
                col = max_col;
            end
        end
    end
end
```

```
Not enough input arguments.

Error in simpleGameEngine (line 31)
            obj.sprite_width = sprite_width;
```

Published with MATLAB® R2020b