F.2 Computer Literacy
Coding with App Inventor 2 - Term 2 Assignment 1

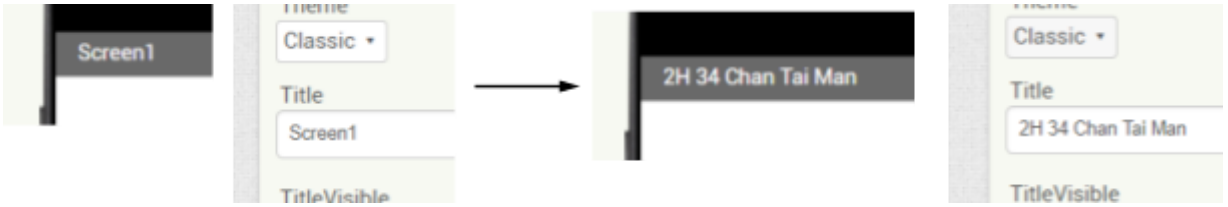
Project idea:

Build a mole-mash type game. Different images randomly appear for short times and when the player touches them, scores are either increased or decreased according to the kind of image touched.

Requirements:

1. Create a new project and name it "molemash_***class_class no***", using your actual class and class number.

2. Check that the screen1 title has Your class, class no. and name



3. Put a large canvas on the screen. This is the main area of the game. Set a suitable background colour (or upload a picture of grassland as the background if you like)

4. Add 5 image sprites on the canvas and name them *hole1*, *hole2*, … *hole5*. Set a fixed size of 80 x 80 pixels for each of them. Distribute them evenly over the canvas area and uncheck their "Enable" properties (so that they will not respond to touches)

5. Upload the given two graphics files *hole_close.png* and *hole_open.png* to the project. Set the picture of the sprites *hole1*, *hole2*, … *hole5* to *hole_close.png* initially.

6. Add another 5 image sprites and name them *mole1*, *mole2*, … *mole5*. Set their sizes to 40 x 40 pixels **and Z property to 2.0** (such that they will never be blocked by the hole sprites). Position them over the different holes such that it looks like they are coming out of the holes. (*mole1* should be over *hole1*, *mole2* over *hole2*, etc)

7. From the emojipedia website (https://emojipedia.org/), **download 3 lovely animal icons** from the "Animals and Nature" categories and name the image files as *good1.png*, *good2.png* and *good3.png*. When the player touches these images, scores are awarded.

8. **Download another 2 scary icons** from the "Halloween" category and name the image files as *bad1.png* and *bad2.png*. When the player touches these images, scores are deducted.

9. Make a dictionary and store it into a global variable *score_dict* for the look-up of score to add for the different image names. Set different positive values for "good1.png", "good2.png" and "good3.png". Set different negative values for "bad1.png" and "bad2.png". The dictionary will be looked-up for the score to add when the player touches an image sprite. **(Follow the demo by your teacher to see how to make and use a dictionary)**

10. Add two clocks for the control of random appearance of these images. Name one clock as *ShowTime* and the other clock as *HideTime*. Read the notes below on the actions of the timers for the way these images should appear.

11. Add another clock for the timing of the whole game. Name the clock as *GameTime*. It is used for count-down of the time for the game.

12. Add suitable labels to display the score and remaining time for the game on the screen, and a start button to start the game, similar to those in previous projects.

Activities during the lesson:

1. Make a dictionary and store it into a global variable *score_dict*. Use the image names (e.g "good1.png") as key and an integer as the value. The value corresponds to the score to add when the image is touched. Try to include several items to the dictionary.

2. Define a function *score_of_image* that takes an image name as input parameter and return the results of looking up the *score_dict* dictionary using the image name as the key. Return 0 if the image name is not found.

3. Use the generic handler [when Any ImageSprite.Touched] to add the score that is looked up by calling the function *score_of_image* with the Picture property of the *component* as the parameter. The score should be added to the score value label that is used to display the score on the screen.

Try touching the different image sprites to see different scores are added.

4. Make a list of mole1, mole2, … mole5 and store it into a global variable *mole_list*.
5. Make a list of hole1, hole2,... hole5 and store it into a global variable *hole_list*.

Note that *mole_list* and *hole_list* are parallel lists as the elements of the same index are related (mole1 over hole1, mole2 over hole2, etc)

6. Define **Functions and Procedure for the game**

Breaking down the complex program into smaller pieces makes program development easier. You can first make the following functions/procedures and later assemble them into other parts:

Procedure: [to add_score (x)]
    For adjusting the score that is displayed on the score value label. Useful in the generic handler [when Any ImageSprite.Touched]

Function: [to score_of_image (image_name)]
    Return the score by looking up the value in the score_dict using image_name as the key. Useful in the generic handler.

Procedure: [to swap_pic (sprite1) (sprite2)]
    Swap the Picture of the image sprites sprite1 and sprite2. Useful in the [to shuffle] procedure below.

Procedure: [to shuffle]
    For each item in the mole_list, swap its picture (using *swap_pic* defined above) with another randomly picked item within the mole_list. After swapping each of them with another random item, the effect is like the moles have moved randomly to other holes.

Procedure: [to hide_all]

Hide all moles by setting their Visible property to false. Also set the Picture of all holes to "hole_close.png".

Procedure: [to show_one (hole) (mole)]

Show the mole by setting its Visible property to true, and change the Picture of the hole to "hole_open.png".

Procedure: [to show_some (*odds*)]

Go through the parallel lists mole_list and hole_list to determine whether the pair of hole and mole should be shown. For each pair of hole and mole, use [random fraction] (a Built-in block under Math category) to decide whether to show it or not. If the [random fraction] is less than the *odds* parameter, the mole is shown (by calling *show_one* defined above)

Since [random fraction] is a random number between 0 and 1, a higher value of odds means a higher chance of showing a pair. So a larger value of odds will make more moles appearing at the same time in each round.

Notes on the actions of the timers

GameTime

This clock is used to keep track of the game time. Its TimerInterval should be 1000 and the action is just decreasing the value shown on the Time value label.

It should also check if the time value has reached zero. When the time is up, it should hide all the moles, and disable GameTime and the ShowTime timer to avoid further appearing of moles.

ShowTime

This clock should first disable ShowTime timer itself, and then show some of the moles (by calling show_some with a suitable value of *odds*) in each round. To make the hiding of moles more irregular, it should set the HideTime TimerInterval to a random value between 1000 to 1600 and then enable the HideTime Timer.

HideTime

This clock should first disable the HideTime timer itself, and then hide all moles (by calling hide_all) and shuffle the images of the moles (by calling shuffle). It should also set the ShowTime TimerInterval to a random value between 1000 to 4000 and then enable the ShowTime timer.

Finally, there should be a Start button to start the game. The start button should

- reset the score to zero
- set the initial time value (the allowed time) for the game (e.g. 30 for half a minute)
- hide all the moles
- enable the GameTime and ShowTime timers