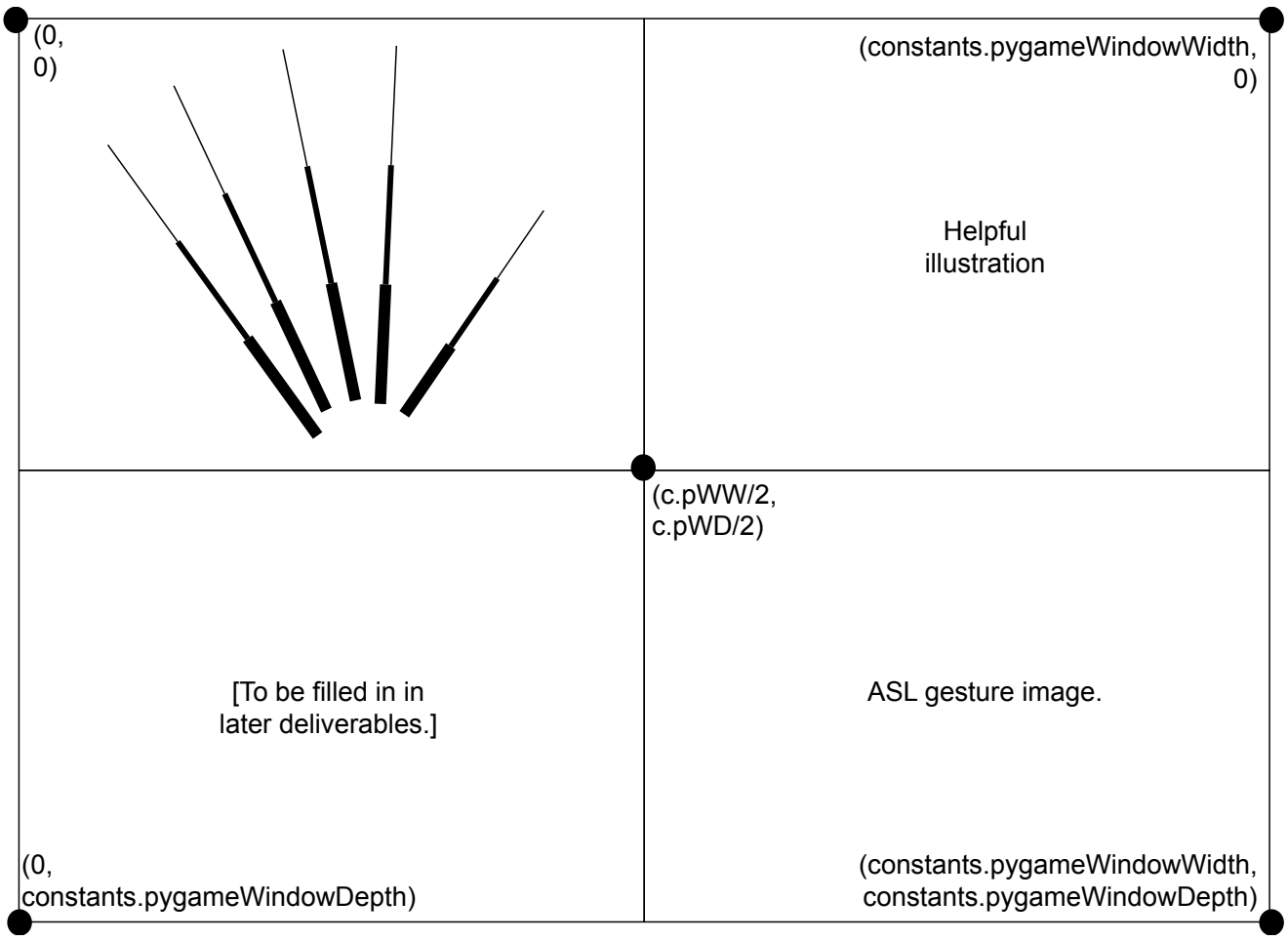

Figure 1: Modifications to the pygame drawing window you will make during this deliverable.

CS228: Human Computer Interaction

Deliverable 7

Description

In this and the remaining deliverables, you will gradually build a system that teaches a user the first 10 ASL numbers. We will assume that the user does not know ASL and has never used a Leap Motion device before.

In this deliverable, you will devise and implement two functions: the first will help the user start interacting with your system for the first time. The second should help the user move and then keep their hand near the center of the device's field of view.

Instructions

1. Imagine that you are a user who has never used a Leap Motion device before. How might this user fail to interact with the device — and your educational software — correctly? Some obvious ones are that they hover their hand outside its field of view. What might you draw to the screen to help the user discover where the device's limits of view are? What might you draw on the screen to motivate the user to hover their hand over the device in the first place?

2. Make a copy of `Del2.py` and call it `Del7.py`. Make sure it is added, committed, and pushed to your git repository.

3. In `Del7.py`, comment out all of the lines that have to do with the classifier. This should lead to code that just draws the user's hand to the screen.

4. We are now going to restructure your drawing window into four panels, arranged as shown in Fig. 1. To do so, alter the scaling function used for drawing the virtual hand so that the hand is 'shrunk' to coordinates in the upper left panel.

   Hint: Note that valid horizontal coordinates for this panel range from zero to `pygameWindowWidth/2`, and valid vertical coordinates range from zero to `pygameWindowDepth/2`.

5. When this is complete and you run your code, you should see the virtual hand drawn in the upper left quadrant of your drawing window.

6. Now, add a visualization to `Del7.py` that the user sees when they start up the program. This visualization should be shown in the upper right of the drawing window. You may choose to draw something using pygame commands, or paste an image there using pygame's `image` command.

   This visualization should motivate the user to hover their hand over the device. If you were to test this visualization with several users (you're not going to yet), you could time how long it takes for a user to bring their hand into the device's field of view, once the program starts running and the visualization is shown. You might average these times across several users, and then try to modify your visualization in such a way that you bring this time interval down across successive rounds of programming and user testing.

   Note: For this affordance — and the rest of the affordances you build into your system — you are not allowed to use explicit text messages such as 'move your hand over the device.' Only graphics, animations or short video clips are allowed. To help you develop affordances, imagine that your system is being used by someone who does not speak English.

7. Now, create a global variable in your code called `programState`. This variable can take on the value 0 or 1, which indicates which of the two states your system is currently in:

   (a) 0 = the program is waiting to see the user's hand.

   (b) 1 = the user's hand is present.

8. By adding this global variable, you are turning your program into a stateful program: the program does different things depending on which state it is in. At this point, restructure your code as follows:

```
(a) while True:
(b)     if programState == 0:
(c)         HandleState0()
(d)     elif programState == 1:
(e)         HandleState1()
```

9. The visualization you just created should only be displayed if `programState==0`. To do this, fill in the `HandleState0` function as follows:

```
(a) def HandleState0():
(b)     DrawImageToHelpUserPutTheirHandOverTheDevice()
(c)     if HandOverDevice():
(d)         programState=1
```

10. You should now add in an `if` clause to `HandleState1` that will set `programState` back to 0 if the hand is no longer detected above the Leap Motion device.

11. Test this new functionality. Your visualization/image should be shown (but the virtual hand should not) whenever a hand is absent. The virtual hand should be shown (but the visualization/image should not) when a hand is present.

    Note: You will have to include the code that draws the virtual hand into `HandleState1`.

12. Next, create a second visualization (or show a second image) that is shown in the upper right panel whenever a hand is detected above the device, in addition to the virtual hand. This visualization should be drawn every pass through `HandleState1`. This visualization should motivate the user to move their hand to some point near the center of the device's field of view. This will require you to detect, at each time step: (1) where the user's hand currently is, (2) where the user's hand should be, and (3) a visual cue to the user about which direction they should move in.

    Hint: You will want to draw different visualizations—or show different images—if the user's hand is too low, too high, too far to the left, etc.

13. If the user is able to keep their hand at the origin for a sufficiently long period of time (you decide what that time span should be), provide a visual cue to the user that they have 'succeeded'.

14. Add another possible value to `programState` such that now

    (a) 0 = the program is waiting to see the user's hand.

(b) 1 = the user's hand is present but not centered.

(c) 2 = the user's hand is present and centered.

15. Add a new function, `HandleState2`, that is called whenever the hand is present and centered. It should only show the virtual hand. This may require you to add some if/then clauses to one or more of the three state-handling functions now.

16. You should now have code that draws...

    (a) a get-started visualization when the hand is absent,

    (b) a guide-for-centering visualization and the virtual hand when the hand is present but uncentered, and

    (c) just the virtual hand when the the hand is present and centered.

17. Check each valid state transition (e.g. no hand to present hand, centered hand to uncentered hand; etc.) by running your code and performing the required action to trigger the transition. Did your code do what it was supposed to do?

18. Now, if the user's hand is centered, pick one of the 10 ASL numbers at random and show it to the user in the upper right panel. Also, show an image of the ASL gesture corresponding to this digit in the lower right panel so the user knows what to do (see Fig. 1).

19. Finally, detect whether the user has made the correct sign or not: this will require you to uncomment the code related to your kNN classifier.

    If the kNN algorithm predicts the same digit 10 times in a row, and that digit corresponds to the one being asked of the user, this will be considered a successful user interaction.

20. Add a new value to `programState`:

    (a) 0 = the program is waiting to see the user's hand.

    (b) 1 = the user's hand is present but not centered.

    (c) 2 = the user's hand is present and centered.

    (d) 3 = the user has correctly signed the current number.

21. When the user correctly signs the current number, state number 3 should be set and enacted. In this state, a 'success' visualization should be presented for a short time period in the top right, a new number should be chosen at random, and then `programState` should be set back to 0, 1 or 2 (depending on the user's hand's current position).

22. Shoot a short video demonstrating each of the above visualizations in action. Upload the video to YouTube and submit to BlackBoard.