Section 7 Report - Tic Tac Toe Game

1. Trace of Moves

Turn	Player	Move (row, col)	Emotions Detected
1	X (bot)	(0,1)	N/A
2	O (me)	(1,1)	row: happy (1), col: happy (1)
3	X	(2,0)	N/A
4	O	(0,0)	row: neutral (0), col: neutral (0); used text override for col (intended col 0)
5	X	(0,2)	N/A
6	O	(2,2)	row: surprise (2), col: surprise (2)
7	X	(1,0)	N/A
8	O	(2,1)	row: surprise (2), col: happy (1)

Result : 0 wins

2. Short Answers

Q: How well did the interface work?
A: The interface worked in that the game ran end-to-end: the webcam opened, a frame was captured on Enter, and the model returned a row and column so the game could place O. In testing, the model often predicted the same emotion (e.g. happy) for different expressions, so the chosen cell was not always the one intended. When that happened, the "text" override (typing "text" then 0, 1, or 2) allowed correcting the row or column so the game could continue. So functionally the interface worked; the main limitation was recognition variety rather than the interface itself.

Q: Did it recognize your facial expressions with the same accuracy as it achieved against the test set?

A: No. On the test set the model reached the required accuracy (e.g. 60% or 70%), but when used live with the webcam it often did not match that. In practice it frequently predicted one class (e.g. happy) for multiple different expressions. There are a few reasons as to why this could have happened. First, the webcam image was a center crop of the full frame resized to 64×64, whereas the training data used pre-cropped. Centered faces at a different scale and setting, so the distribution of inputs ended up different. Second, the lighting, angle, and face size in the room differed from the data set. Finally, the last possible reason was that the model may have learned to default to a frequent class when inputs looked ambiguous. This probably made the expressions more exaggerated.

Q. Lines of code added in `_get_emotion` for emotion detection

A.

```
def _get_emotion(self, img) -> int:
    if getattr(self, '_model', None) is None:
        path = os.path.realpath(_get_model_path())
        self._model = load_model(path)
    img = cv2.resize(img, image_size)
    img = np.expand_dims(img, axis=-1).astype(np.float32) / 255.0
    img_batch = np.expand_dims(img, axis=0)
    pred = self._model.predict(img_batch, verbose=0)
    return int(np.argmax(pred[0]))
```

The helper `_get_model_path()` and the constant _MODEL_FILENAME were also added to the top of the file so that path to the saved .keras file was relative. This helped reduce redundancy in the code overall.