Maximum total 100 points. Part 1 is 50 points and part 2 is 50 points.

You are NOT permitted to post, discuss or work together on the assessment questions with other students in the class or with anyone else. Piazza will be closed for the duration of the assessment period except for private messages to the instructor. Your answers can be as long or as short as you want, but a long meandering answer that incidentally mentions something relevant will receive less credit than a short high-quality answer. Submit your answers all in a single document in response to this assignment on courseworks. You can submit as many times as you want until the deadline. Submissions via email, piazza, or any other means besides courseworks will not be graded.

Part 1 (50 points) Reverse-engineer **your own implementation** of tic-tac-toe from the individual project to produce documentation in the form of *use cases*. The same IA who graded your individual assignment will grade this part of the assessment. There are three subparts, A, B and C.

1A (25 points) Write a set of use cases covering the entire functionality of **your own implementation** of tic-tac-toe from the user (game player) perspective. Use the latest version you submitted (but ignore the database and recovery support for these questions).

You should write a *set* of use cases, not a single huge use case for "Play tic-tac-toe". Every use case should have at least a title, actor, description, and basic flow. Most use cases have triggers, preconditions, postconditions, and alternative flows. Some may also have induced events and inclusions. Make sure to label every step in each flow, including the branch point for alternative flows. If your triggers, events, preconditions and postconditions have distinct subparts, label those too.

1B (15 points) Associate each of the major methods in your code with the particular parts of the use case(s) they implement. Omit any methods concerned entirely with the database and recovery. You can also omit constructors, getters, setters, utility functions, etc. *unless* that is the only part of your code that implements a given part of a use case. That is, every part of each use case from 1A needs to be associated with some code that implements it – otherwise it was not implemented and thus should not be included in your answer to 1A.

One way to make sure you have covered everything is to walk through every method in your codebase and label it with respect to which part of your use cases it implements or decide it is not applicable (e.g., testing or recovery code). Keep in mind that some individual method might implement multiple parts of a use case and some parts of a use case might be split among multiple methods. Some methods might be reused in multiple use cases.

1C (10 points) Knowing what you know now, what would you have done differently when you implemented your original codebase for the first individual assignment?

Part 2 (50 points) Recall our breakout room exercise with a drone that flies around the classroom and uses its sensors together with AI to spot students who are not paying attention during the class. Here you will design a Zoom API and sample plugin to similarly inform the host which Zoom meeting participants are present but not "paying attention". Do not use the camera and microphone, but instead assume every Zoom user is supplied (for free!) with their choice of Windows, Mac or Linux laptop with a Soli chip built in. Students are required to use these laptops for Zoom class sessions. The Soli chip is described at https://atap.google.com/soli/ and https://atap.google.com/soli/technology/. There are four subparts, A, B, C and D.

2A (25 points) The only interface to the Soli device drivers is a REST API provided by the Zoom client that comes with every Soli laptop. Design this REST API with at least two methods such that a third party could develop their own paying-attention plugin. Your API should be presented similarly to the real Zoom APIs as documented at https://marketplace.zoom.us/docs/api-reference/zoom-api (expand a couple of the categories and look at a few pages from the menu listed below "ZOOM API" on the left hand side). That is, for each API method you should write the REST-METHOD and endpoint path, a prose description, request parameters (if any), request body (if any), and responses. You can omit prerequisites, scopes, rate limit label, authorization and test request.

Assume the user cannot turn off Soli except by shutting down their laptop, in which case they are obviously not paying attention. We will not be picky about what the Soli device or its drivers can really do - invent anything that seems plausible. All the AI has to be in the plugin implementation and its backend server(s), do not include any functionality in your API like "is the user of this laptop paying attention?" Do not waste time learning the real Zoom APIs, which will not be helpful, just mimic the structure of the endpoint documentation for your API methods.

2B (15 points) Write a set of two or more user stories, each with conditions of satisfaction, for a paying-attention plugin for Zoom clients. Assume whatever AI seems plausible, which could run in the cloud not necessarily on the Soli laptops.

2C (5 points) Describe how you would conduct acceptance testing on the REST API of 2A.

2D (5 points) Describe how you would conduct acceptance testing on the Zoom client of 2B.

This is the end of the assessment.