1) What can we do on a computer than we can't do on a printed board?
On a computer we can quickly display new views, refresh the view, alter the view, or close the view of an application. Additionally we can quickly modify the data and have the view update the model based on the new data. A computer also allows for direct user input and can provide feedback based on the user's input.

2) What is a computerized opponent? Remember, not everyone is an expert. What are its objectives? What characteristics should it possess? Do we need an opponent or opponents?
A computerized opponent is an entity that plays one side of a game. Generally, it is an algorithm, whether static, an AI, or has some form of ML capabilities, that will play against a human player. This opponent is expected to play LIKE another human player, making smart moves to try to win the game. A smart move is a move that follows at least one of the following characteristics (however this list is not exhaustive): has the computerized player closer to winning the game, weakens the opposing players position, strengthens the computerized player's position, follows some kind of strategy (learned or innate).
For Connect4, we only need a single opponent. This opponent however could have multiple difficulty levels for it. If the designer wants to get fancy they could have multiple players playing on the same board, however I am not sure how this would impact the games mechanics.

3) What design choices exist for the Interface components? Colour? Font? Dimensions of Windows? Rescale-ability? Scroll Bars? ….
For connect4, the entire board should be displayed at once so the user can see what the complete board state looks like in one glance. This should also be resizable from a minimum to however large the user would like. Two assumptions is that the board has to take up a minimum resolution for the board to be completely visible, and that the user does not have a ridiculously large resolution so that memory becomes an issue.

The colours of the tokens should also be distinctly different so that players can tell which tokens belong to which player.

Fonts should contrast the background in a readable format (soft white on black, not purple on lime green).

4) What are the advantages and disadvantages of using a visual GUI construction tool? How would you integrate the usage of such a tool into a development process?

The advantage of a visual GUI construction tool is that the constructions of the View elements can be rapidly developed and pieced together, allowing the CAD to take care of most of the small details. An example of this is in android studio where the xml file is generated as elements are placed on the view. In addition, it also can link buttons to trigger certain actions in the controller class code.

One drawback however is that the developer limits themselves to what the CAD tool can actually create. Providing that the tool isn't too limiting, generally the increased productivity is an acceptable trade off.

The integration of the tool should come at a time when the View class is starting development. The contract of what the View looks like and does should already be decided so that it does not interfere with the development of the controller class. The View and COntroller class should try to remain as independent from one another other than method names and how to interact with one another in an agreed upon manner.

5) What does exception handling mean in a GUI system?
Can we achieve consistent (error) messaging to the user now that we are using two components (Ruby and GTK3 or other GUI system)?
What is the impact of the Ruby/GTK3 interface on exception handling?

In a GUI system there are now two types of error handling. The traditional error handling in that catching when something goes wrong in the Model class (IOException, StandardError), in other words the actual logic. The second kind of error handling is now with the View. This error handling is if something fails with the element in the view class. This could take the example of an element failed to initialize properly, resizing broke the view, an element wasn't properly configured to handle a click, or the View tries to call a method that doesn't exist from the controller.

To pass error handling from the model to the view, the model should pass the important data (title, message body) to an error_handler method in the controller class. This controller class could then construct a GUI element to pass to the view class where it could then display the error to the user.

6) Do we require a command-line interface for debugging purposes????? The answer is yes by the way – please explain why.

This is required because the View may not be accurately showing the actual data that is currently in the system. (The data that the model currently has in its memory).

A command line interface is also required for debugging to allow to give developers feedback on the system when an action is attempted (note attempted instead of completed). This can work like a logging system and is able to show what the data looks like in its "raw" form. The form form being like a string representation.

Creating this kind of debug log in the actual view would be tedious, may be error prone, and is not worth the time required.

7) What components do Connect 4 and "OTTO and TOOT" have in common?

How do we take advantage of this commonality in our OO design?
How do we construct our design to "allow it to be efficiently and effectively extended"?

Connect4 is about making a continuous line of 4 tokens of the same colour. "OTTO" and "TOOT" differ by matching a pattern of characters. How we can take advantage of this commonality is by using pattern matching (read RegEx) to check the board. If a pattern "TOOT", "OTTO", or "RRRR" (4 red tokens in a row like in the classic game.) is detected, then the last player to play a token won. This means if we look for a pattern, we can set our pattern to be of a set of arbitrary tokens (or representation of characters) of arbitrary length (subject to game board restrictions).

After a token is placed, the game should check if a pattern has been made from where the token was dropped. It doesn't make sense to check the entire board as most of it will be empty (beginning of the game) or unchanged (late in the game).

8) What is Model-View-Controller (MVC, this was discussed extensively in CMPUT301)?
Illustrate how you have utilized this idea in your solution. That is, use it!
MVC is a design pattern that describes how different layers of the GUI should interact with one another.
Model: Contains the logic for each entity. The Model contains the data inside of it as well as required methods to modify or access the data that is contained inside of it.
Controller: Acts as the layer in between the Model Class and the View class. The controller class is responsible for handling actions between the view class and model class and vice-versa. When it receives data from an external source (database, internet, etc.) it calls the View to modify its contents to represent the event that had just happened.
View: Contains the logic used to construct a GUI, this includes all of the elements of the interface including but not limited to: buttons, tabs, options, radio buttons, text entry spots, registering when something was clicked. Based on what was clicked, the view then calls the appropriate controller method to handle the request.

9) Different articles describe MVC differently; are you using pattern Composite?, Observer?, Strategy? How are your views and controllers organized? What is your working definition of MVC?

For this assignment, we believe that the Observer Pattern is best suited. Here we have our Connect4 model, implemented as a NxN array. A View that visually represents the Connect4 board, and a controller that communicates between the view, and the model. Here the View contains arrays of listeners that represents interactable objects on the GUI. The view will notify the controller, whenever an event is detected, and the controller will make changes to the Model. Also, the Model will be an observable, that the graph monitors for any change to be communicated to the View.

10) Classes start life on CRC cards or a competing notation. Provide a full set of CRC cards produced by your group for this problem. These cards must be supplied as part of part 1.

11) Iterators – are they required for this problem? Fully explain your answer!

Iterators could be used for this problem. The iterator would be to go through the list of players, going to the next player once the current player took their turn. After all players took their turn, the list would repeat itself before going back to the top of the list. This loop would be broken as soon as a winner has been declared.

12) Explain your strategy for testing the GUI component of the system. How are you going to test that your system is Usable?

I think the best way to test the GUI system is to "use it" ourselves. The reason for this is that not only does it allow us to test that the component is working as expected, but we also get to test whether the system is pleasing/intuitive to use. Ideally, this testing should be done by people that are not part of the development process (to get rid of any preconceived bias) and should be done by different people. However, due to the restriction, we will just do it ourselves.

13) Colour (color) blindness (colour vision deficiency, or CVD) affects approximately 1 in 12 men (8%) and 1 in 200 women in the world. In Canada this means that there are approximately 1.5 million colour blind people (about 4.5% of the entire population), most of whom are male. Explain how your design accommodates this user group and how you will test this accommodation to ensure that the final implementation meets this objective.

According to the National Eye Institute (department of National Institutes of Health) (https://nei.nih.gov/health/color_blindness/facts_about) There are 3 main types of colourblindness: Red-Green Colour Blind, Blue-Yellow Colour Blind, and Complete Colour Blindness.

To accomodate each form of colour blindness, One option is to pick colours for each type so that the user does not have a difficult time to distinguish the colours. This would form different colour settings the user could choose from.

For Red-Green: The 1st main colour should be red and the 2nd main colour should be blue as blue is not affected by this colourblind condition. The red however will appear either mild/faded to brown or beige.

For Blue-Yellow: The 1st main colour should be red and the 2nd main colour should be green as it is not affected. Neither colour should become distorted when viewed by someone who is Blue-Yellow colourblind.

For Complete Colour blindness, as someone who only sees in greyscale Black and WHite are the most obvious choices. This however requires other secondary colours (like for background) to be carefully chosen in greyscale.

An alternative option if the above is too tedious, is to write a letter on each token in black or white (looking for high contrast with background of token) such that the user could instead focus on reading tokens instead of identifying the colours.