ECE 421
Project 4

Christina Ho
Lindsay Livojevic
Jonathan Ohman

1) What can we do on a computer than we can't do on a printed board?

Computerized game board can be reused an unlimited number of times. Resetting the game to its initial state is much quicker. Not limited to physical limitations like table space or board size. Can have an enemy AI to play against. Can play through a network so your opponent doesn't have to be physically present.

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 a program that simulates a player of the game, against which the user can compete. Its objective is to win a given game against its own opponent. It should understand the rules of the game it plays, and calculate the most prudent immediate strategy given the current state of the game pieces. For a one-player game of Connect 4, one opponent is needed.

3) What design choices exist for the Interface components? Colour? Font? Dimensions of Windows? Rescale-ability? Scroll Bars? ….

For our project we have decided to represent the board with a Table whose slots will be filled appropriately to represent our tokens. Buttons placed above the Table will be used to decide which column to place tokens in. Error messages will likely be displayed in a popup. The menu/settings view will make use of radio buttons.

Font, colour, and general aesthetics will be kept to a minimal look to increase readability of the user interface.

The dimension of the game window should default to a given resolution, however it is likely that different users will be using machines with different resolutions and displays. We will attempt to include the ability of rescaling the game window to improve user experience. Hopefully this will be possible due to the simplicity of the planned components.

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 using a visual GUI construction tool would be the ability for the designer to see exactly how the user will see and interact with the finished GUI, and let that inform their design decisions. The disadvantage is that auto-generated code from the visual construction may be unfamiliar to the developer and present challenges in modifying it.

Integration of a visual GUI tool should not be a difficult task as long as our code adheres to MVC structure and we have a good understanding of how the data is manipulated.

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 GTK2)?
What is the impact of the Ruby/GTK2 interface on exception handling?

Exception handling in a GUI system requires that the GUI system continue functioning despite an error. In other words, the GUI should not stop working due to bad input or bad event handling. We should also probably still catch and handle the exception for debugging purposes.

Yes, if we wanted all errors to look consistent we could create an overall rescue clause and display the error to the user in popup for example. But more than likely, we will translate specific errors into user friendly formats.

The interface between Ruby and GTK2 could pose a number of problems in the signal handling. For example, every button, menu option, or interface object would require manual testing from the user/developer to discover missing functionality or misaligned signals. To protect our Ruby/Gtk2 application we can create our own function that automatically wraps the desired Proc in signal_connect with a rescue clause. (Eg. applyEventHandler(widget, event, &handler).) We will then exclusively use this function to apply any event handlers. This would prevent event handlers from ever crashing the GUI. Then we only have to worry about the GUI code crashing itself.

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

Yes, to assist in making debugging more accessible. There are also some elements that we as developers should know about that should not translate into something the user sees and can be displayed on the command-line interface.

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"?

Connect 4 and "OTTO and TOOT" have the same number of players and the same gameplay. The only difference is in the win conditions, which can be represented in a common way. A player wins if a line exists that matches a certain pattern of their own pieces and their opponent's pieces (Connect 4: { own, own, own, own }, OTTO/TOOT: { own, opponent, opponent, own }). To implement both Connect 4 and OTTO/TOOT, only that pattern must change, with all other objects being common. Any other game that follows these criteria can then be implemented.

8) What is Model-View-Controller (MVC this was discussed in CMPE 300 and CMPUT301)? Illustrate how you have utilized this idea in your solution. That is, use it!

Model-View-Controller is a code structure that cleanly distinguishes the user interface from the back-end code. The data and data manipulation methods are handled in the Model classes, and the GUI code is in the View classes. The Controller methods are to be called by the GUI's event listeners, and in turn call Model methods, providing a connection between the front-end and back-end code.

Our MVC works as such:

- View - Our view will be represented in two parts, the menu and connect four game board. There will two implementations of the view: the gtk gui, and the cmd line interface. The view will have a single entry point through MenuView which will accept options to specify: the BoardView type, and the menuController type. The menu implementations and board implementations should be interchangeable. The BoardView is initialized by the menuView, it supplies the boardView with the desired type (gtk or cmd) and an initialized boardController. The views can potentially accept any valid implementation of their controller interface.
- Controller - The controllers are broken down very similarly to the view. There are two types of controller, menu and board, which correspond respectively to the views menu and board. Both controllers have a single entry point which defines the interface for the controller type and allows the programmer to specify which implementation of the controller the view would use. The view will register controller functions to certain events and will use the controllers' responses to update the view.
- Model - We have a two main Models: settings and game. Settings represents the settings for a game (rows, cols, victory mode, number of players, etc). The game model accepts a settings object as one of its attributes. Additionally, the

game model contains a 2D representation of the game board, and variables to keep track of whose turn it is, if there is a winner, etc.

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?

- We are not using observer as the model because it does not directly tell the view of its state changes. The changes are instead returned to view (as the complete model) from each call to the controller.
- We are using the strategy pattern as each view piece is swappable with a corresponding view piece. Each view piece also accepts any controller that meets its respective interface, making different implementation of controllers also swappable.
- We are using the composite pattern as the view only has one entry point for updating its entire appearance.

Our working definition of MVC is:
There is a common interface between each view and and corresponding controller. This allows any view that uses the interface to use any controller that implements the interface. The view uses controller functions in its events, and uses the returned values to update the view. The models are very simple and know nothing about views and controllers. The Controller knows how to manipulate the model and the View knows how to display the model.

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.

See bottom of document.

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

Iteration would be required to check for victory conditions. We will need to iterate over the board to check if a victory condition has been met (OTTO/TOOT/RRRR/BBBB). Potentially, we could minimize iteration by only checking if a victory condition exists because of the most recently placed token. That way we wouldn't have to consider any tokens more than 3 away from the current one. We could also start with closest neighbors and only continue checking in the neighbor's respective direction if the victory condition is met.

12) What components of the Ruby exception hierarchy are applicable to this problem, etc? Consider the content of the library at:

http://c2.com/cgi/wiki?ExceptionPatterns Which are applicable to this problem? Illustrate your answer.

- Interrupt - we might need to catch ctrl+Z and ctrl+C in case those used while using the cmd line interface to clean up properly.
- ArgumentError - This is thrown from placeToken when the user attempts to place a token in a column that is full. Should be caught and used to notify user of their error.

We will probably be catching only the exceptions that we can handle and/or know that we will be handling, such as the above. Because of the MVC model, we will also need to let errors propagate to an appropriate higher level, rather than try to handle specific exceptions at lower levels.

CRC Cards

MenuView	
To present a the user with options to configure a connect4.2 game. Once the user completes the options they can then choose to start the game.	MenuGtk MenuCmd MenuController BoardView SettingsModel

MenuGtk	
An implementation of the MenuView. Provides a graphical user interface. Built on gtk.	MenuController BoardView SettingsModel

MenuCmd	
An implementation of the MenuView. Provides a cmd line user interface.	MenuController BoardView SettingsModel

MenuController	
To be used by the MenuView. The MenuController will manage a SettingsModel object on behalf of MenuView. It will attempt to complete requested actions on behalf of the view, on success it will return the updated model to view.	MenuDefaultController SettingsModel

MenuDefaultController	
An implementation of the MenuController. Default implementation, just simply modifies the SettingsModel object.	SettingsModel

BoardView	
To present a visual representation of the current Connect4.2 game. It also provides an interface for the user to interact with the game.	BoardGtk BoardCmd BoardController GameModel

BoardGtk	
An implementation of the BoardView. Provides a graphical user interface. Built on gtk.	BoardController GameModel

BoardCmd	
An implementation of the BoardView. Provides a cmd line user interface.	BoardController GameModel

BoardController	
To be used by the BoardView. The BoardController will handle user requests such as placeToken(column). It will attempt to complete the requested action on behalf of the view, on success it will return the updated model to view.	BoardLocalController GameModel

BoardLocalController	
An implementation of the BoardController. Simply modifies the GameModel object.	GameModel

SettingsModel	
This model represents all the settings for a game a game of Connect4.2 that will be unchanged for the round (eg. rows, cols, victory mode, number of players, etc).	

GameModel	
The game model represents a current game of Connect4.2. It contains a 2D representation of the game board, and variables to keep track of whose turn it is, if there is a winner, etc. It will also contain a SettingsModel object as it contains the unchanging options that are tied to the game. It also uses the VictoryModel to make use of different victory types (normal/otto).	SettingsModel VictoryModel

VictoryModel	
Provides a model for the different types of victory (normal/otto). The program will be able to select which kind of victory it would like to be represented by their victory object.	VictoryNormal VictoryOtto

VictoryNormal	
An implementation of VicotryModel. Uses the normal victory conditions of Connect4.	

VictoryOtto	
An implementation of VicotryModel. Uses the OTTO/TOOT victory conditions.	