Assignment 4
Group 1
Part 1 Questions

March 20th, 2018

Bianca Angotti (bangotti 1417422)
Andrew McKernan (amckerna 1432175)
Arjun Kalburgi (kalburgi 1388774)

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

Computers can provide drastically different user interfaces and experiences based on how they are programmed and how the viewer responds to it. Computers allow for different takes on traditional games as well, since traditionally set variables, like board size and play pieces (number of players), can be changed to be anything. Game history can also be kept in memory to support reviews and undos of moves. Games can also be made such that users can play against the computer. A traditionally 2-person printed board game can now be played as a single person against a computer that can vary its difficulty levels.

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 when logic is programmed in order to serve as a user. This logic can be built for 2-player games so that one of the players (or both) can be non-human. The objective of such a program is to provide the user with a 2-player experience while being the only player available.

Computerized opponents should be able to provide some level of difficulty during play, this difficulty level can be set depending on the algorithm. Most games use a minimax algorithm with a relatively simple heuristic function in order to rank outcomes. The level of depth of the minimax search will be related to the level of difficulty of the computerized opponent.

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

UI and UX design have now grown into large industries and are strongly considered in the agile workflow. It is now believed that all parts of the interface are needed to bring about the desired experience for the user. This includes colour, text hierarchy, copywriting, object placement, animations, and more. With hundreds of products vying for user attention, design today is often made to immerse users, grabbing attention with all available interface components and business logic. It is important to keep in mind that not all users are able to see color or hear sound, and some are unable to use mouses or keyboards as efficiently as able-bodied people. Hence, design choices should keep in mind those who interact with computers in different ways.

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?

A GUI construction tool allows for a GUI to be built fast and efficiently, without the difficulty of considering all the design complexity mentioned in question 3 for each and every product. For many cases, having and using a default style is much more efficient and sufficiently effective. However, there may be some disadvantages in that a GUI can often be slower or more limiting to a user, especially if the user is highly skilled in the program they are attempting to use. Consider the git GUI versus the command line interface - this GUI makes things easier for users grasping git for the first time, but can be quite limiting to highly skilled command-line users.
Tools such as these are usually integrated via the IDE, such as in Android Studio. For the purpose of this assignment, we will be working with Glade and GTK3, as they provide a strong basis to build off of. By using pre existing libraries, it speeds the process of development, and since these are widely used and updated libraries, we have a general guarantee that these products work well by the support of the community.

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?

A GUI is not a place for error and exception messages, and in most cases users should not see these. Game execution messages should be handled differently that user input error messages. The GUI construction tools may contain specific components for showing the user both of these types of messaging, perhaps in a log (Android Studio) or perhaps via a notification (Android Beta bug reports). The GTK3 interface allows us as programmers to ensure this distinction without too much additional programming. When appropriate, error messages can be alerted to the user via the GUI. These messages are presented in a more packaged way, stripping down full error messages used for debugging purposes. These messages are important for developers in the process of making a product (usually via command line and stored in a log) but can be reduced for the user's view.

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

The command line is used for debugging as a standard, as it's a much more testable format. Removing the GUI allows for more direct access to the logic of the code, and is thus much more flexible for debugging. Our project implements the GUI on top of the logic of the game, thus we have this flexibility. As well, if there is an issue with the GUI itself, it may not be able to be debugged simply by moving through the GUI. A command line interface is critical for printing out debug information that gives insight into the source of a given issue that is occuring.

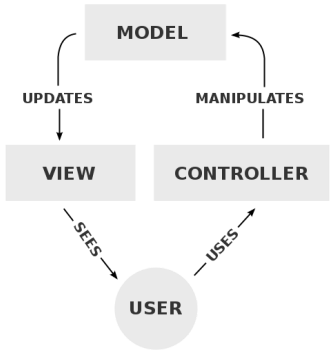
7. **What components do Connect 4 and "OTTO and TOOT" have in common?**
   How do we take advantage of this commonality in our OO design?

How do we construct our design to "allow it to be efficiently and effectively extended"?

Connect 4, "OTTO", and "TOOT" all have similarities in that they take place on a similar board, and all involve placing tokens on the board in a specific order while attempting to prevent your opponent from doing the same thing. This makes the potential advantages of object oriented design clear - the tokens placed by a user can all be fundamentally treated as the same thing (perhaps a "Token" class) but the conditions for victory for each game type will be different (implemented in classes inheriting "Token", such as "TOOTToken", "OTTOToken", or "Connect4Token"). This makes for a simple design that is extendable as well - if we ever want to add on a different game type using this format, all that needs to be done is to add another class inheriting Token with slightly different win conditions.

8. **What is Model-View-Controller? Illustrate how you have utilized this idea in your solution.**

Model View Controller is an architectural pattern often used in UI that allows for an interconnected set of logically separated parts to be easily developed alongside each other. Below is a diagram of the basic MVC model:



As seen, the system is broken into three components: The View, the Controller, and the Model. The User observes the View, and makes requests or commands to the Controller. The Controller makes changes to the Model, and the Model finally updates the View to reflect these changes. This is done primarily to enhance maintainability - in theory, the Controller, Model, or View can be updated without having to modify aspects of the others. This allows a programmer to modify aspects of the Model or Controller without having to change at all what the User sees in the View, or conversely, change aspects of the View visible to the User without actually changing the implementation of certain routines in the Controller or Model. This makes for a very modular, expandable, and maintainable architecture.

Models: Player, Board, Game, AI
Views: MenuGUI, GameGUI
Controller: Driver

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?**

In our understanding of MVC, as well as what we have read in various articles, MVC can mainly be thought of as a rough combination of Composite, Observer and Strategy design patterns. Our pattern that we are going to be focusing on is the Observer pattern. The View aspect of MVC acts as the "Observer" as it will be updated when the Model (Observable) calls a notification, telling the Observers that it has changed.

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.**

Game	
Responsibilities	Collaborators
- Keep track of current players of the game - Execute moves of players onto the board - Check if a move made results in a win for one of the players - Set game type, dimensions, and number of players	- Board (used by Game) - Player (used by Game)

Board	
Responsibilities	Collaborators
- Clear all pieces from gameplay - Add a token in a certain placement on the board and store that information - Provide information if there are open spots on the board and if the board is full	- Game (used by Board) - Gui (used by Board)

GUI (menu/game)	
Responsibilities	Collaborators
- Display game/menu to the player in an easy to read and play format	- Board (used by GUI) - Player (used by GUI)

Player	
Responsibilities	Collaborators
- Place tokens in their desired position on the board	- Board (Player modifies Board) - Game (uses Player) - Gui (makes and uses Player)

AI	
Responsibilities	Collaborators
- Make a move that it calculates to be the best in response to a human's move	- Player (AI is a Player) - Game (uses AI) - Gui (makes and uses Player)

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

The game board is an array of arrays, thus iterators could be used for the check_win. The AI also needs to iterate across the board in order to find an appropriate place to drop it's tokens as it attempts to win. As much as it may seem a bottleneck, iterators are still required to analyze solutions and standings. It is not possible to evaluate all potential win conditions on a board at a given time without having some form of iteration take place (finding a valid combination of OTTO or TOOT). As well, since this is a game structure with two players, whether AI vs Human or Human vs Human, we can iterate through the players.