ECE 421 Assignment 4 Part 1

Winter2019_Group4:

Nathan Klapstein (1449872)

Tony Qian (1396109)

Thomas Lorincz (1461567)

Zach Drever (1446384)

Table of Contents

Description

Traditionally, many families have spent their evenings playing board games; today, one potential money making strategy is to computerize these games and to sell them to a new generation. To increase the probability of capturing a reasonable market share, our electronic media designer continually ponders the question: “What additional characteristics or features could a computerized version possess that could differentiate it from the original tactile version?” The project is to produce a computerized version of the game: Connect4. One obvious answer, to the question of adding features, is a computerized opponent(s) for individuals lacking playing partners. A second obvious answer is to add variations on the basic theme; for example, as well as playing Connect4, the system should also play OTTO and TOOT. This new game has many similarities to Connect4, players now place “O” and “T” counters; and one player wins by connecting 4 counters spelling “OTTO” and the other by connecting 4 counters spelling “TOOT”. Our company expects to produce a number of games upon this basic theme, and hence, you are required to design your game to “allow it to be efficiently and effectively extended”; a quote from our CEO. Producing more than two games, at this point, would obviously be a great advantage. Our company expects (you) to produce a number of games upon this basic theme, and hence, you are required to design your game to “allow it to be efficiently and effectively extended”; a quote from our CEO. And essential for Assignment 5!

As stated above, Assignment 4 is to produce a computerized version of the game(s): Connect4 and its variations, both specified (TOOT and OTTO) and unspecified (“to be efficiently and effectively extended”). This new version should seek to exploit the differences between this new implementation medium and the traditional or original medium.

Design Questions

1. **What can we do on a computer that we can’t do on a printed board?**

   In general, a computer is a collection of modular PCB components that work together to provide a general purpose computing system. A PCB is usually a specialized electronic circuit that can take in inputs and produce outputs. PCBs usually do not have a visual output (e.g. HDMI or DP). Most personal computers are differentiated by this quality. They are designed to handle a wide variety of user tasks and applications are provide a convenient way to visualize the data underneath (e.g. a screen).

   With the ability to abstract the data into a visual representation (a User Interface), user’s will have an easier time manipulating and visualizing their personal information. As well, this gives developers the challenge of effectively presenting an application to a user. An advantage of this abstraction is the changeability of a UI over a PCB. While the user is not able to change how data is stored in a simple PCB with persistent memory (at least, they shouldn't do this), the user interface is adaptable and can be focused so that primary use cases are presented noticeably and with intuitive controls. Thus, the advent of UIs for personal computer applications allowed for the widespread adoption of using computers for what was previously performed on paper.

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 algorithm that is designed to play a game against a human player. A well-designed computerized opponent should be able to make intelligent decisions that make it fun to play against. A challenge in creating computerized opponents is programming their difficulty level.

   For the connect 4 application, only one computer opponent should be programmed. Though, it would be interesting a potentially useful to allow a computer to play another computer. This way, the gameplay and computer strategy could be tested through many simulated trials.

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

   We should allow resizing of the application window to the point where all elements can still be displayed within the window without the usage of panning, or a scrollbar to change viewpoint.

   In general, it is best to avoid needed scrollbars for a single-page app or game if at all possible. If everything is not able to fit on the screen, that would be a sign of poor layout design.

   Font should be readable, stylized to match the tone of the game, supported on multiple operating systems, and consistent. If a playful, non-serious tone is expected for the application, then the font should reflect this.

   Similar to font choice, colours should be chosen to reflect the tone of the game. As well, colour blindness will be taken into consideration when choosing the palette of the game.

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?**

   Advantages:
   - Easier to visualize constructed components as direct visual representations are presented
   - Usually in highly integrated and controlled projects this can lead to accelerated development times.
   - Less programmatic; more intuitive

Disadvantages:

- Custom functionality or GUI components may not be supported by the visual GUI construction tools
- Usually locks-in development to one type of GUI API or library as the visual GUI tools are built off it exclusively.
- Not always free
- Harder to obtain a direct to code solution / auto generated code is now present leading to the developer to be at the mercy of the generated solution.
- Visual GUI construction tools are usually workstation resource hogs, comparative to text/code editors.

How would you integrate the usage of such a tool into a development process?

It depends on the process and the capabilities of the tool. For example, Android Studio has an integrated GUI construction tool that provides an intuitive interface for beginners and an accompanying code window to allow more experienced developers have more control. Because the two are so well-linked, most developers will find it easy to use both to aid in their development process.

In general, so long as the GUI construction tool limits its scope and doesn't try to do too much, it can be an excellent way to separate the visual/design concerns of a GUI from the logic/programming concerns which could be handled by a different tool.

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?**

Exception handling in GUIs requires careful consideration of user experience. Game applications should always try to handle errors internally as much as possible. Since the goal is to emulate a game that exists physically, the user experience of the application should not have pop-ups and visible error codes (because these don't happen in real life). Of course, error with how the game is rendered can lead to some visual glitches, but most people who play games are willing to accept these types of errors.

To interface with the GTK3 framework, our exception handling will need to wrap the exceptions that are thrown by GTK3. This way, we can divert control of exceptions and logic to the appropriate components of our application (e.g. the Model classes). As well, as we become more familiar with GTK3, we will learn about the effects of error propagation and whether we should use custom components instead of built-in components based on default behaviours.

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

Yes, having a cli interface allows for easy testing via automated CI systems such as Travis-CI. Lacking a cli interface would make adopting such automated CI systems difficult. Lacking automated ci systems is a bane on projects as software validation becomes a much more manual process. Especially with a GUI, having a headless core is important because it allows us to simulate the running of the app without having to manually click components with a mouse. Debugging using physical devices is often inconsistent and difficult to log and diagnose.

Adopting a MVP architecture for our application will help in this regard because it allows the application to be headless. All data/state interactions are abstractly initiated by events (controlled by presenters). Thus, the command-line can be treated as a view the same way that a windowed GUI is. The model is unaffected by the input choice, it just responds to user events.

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”?**

   Both game mode components consist of O and T objects to construct a pattern that leads to a victory condition. These objects can be reused to construct the OTTO and TOOT gamemode components, thus, limiting code duplication.

   The TOOT, OTTO game mode components follow similar premises of victory, but, differ simply on the order of the objects required to create a pattern that leads to a victory condition. Thus, both the TOOT and OTTO game mode components could be subclasses of a generic TO base game components, and within the subclass definition they only implement a required order to obtain a victory condition.

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!**

   Model-View-Controller is an architectural pattern that is effective for making GUIs because it is maintainable due to its separation of concerns. However, as discussed in CMPUT 301 and confirmed by professor Hindle and his PhD students, Model-View-Presenter is much more understandable and scalable compared to MVC. We will be using MVP as the architecture for our project.

   In MVP systems, the models represent the state and business logic of the application. They are the only components that understand how to update the data. Thus, the representation and manipulation of data remains isolated from other components. Views are components that know how to receive user interaction (clicks, key presses, text input etc.) and, optionally, how to draw themselves to the screen. This allows views to be incredibly unintelligent. Keeping views unintelligent means that the views are not tied to data and can therefore be removed entirely without breaking any of the system logic. Finally, presenters are adapters that connect models and views. A presenter will subscribe to events that are triggered by user interactions with views.

Once a presenter is notified of an event, it can prepare data for use in the model (input sanitization, for example) and then call a method in the model. Once the model has updated state/data based on the user interaction, it triggers an event. One or many presenter will be listening for this model event and will tell their views to redraw as needed (likely by inputting the new model data into the view drawing functions).

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?**

   Composition, Singleton, State, Observer and Strategy pattern are all patterns that can and should be used to model this problem. Composition can be used to model the board and pieces that are used to compose the board. The singleton pattern can be used to keep track of game state. This can be used to enforce only one game taking place at a time. State pattern can be used to keep track of the state of the game and calculate whether the game is in a winning state. Observer can be used to update views as the game state changes. The Strategy pattern could be used to select an algorithm for the computerized opponents to use during play. If multiple algorithms for the selection of placing pieces is written, then the selection of algorithms could allow toggling between 'easy', 'medium' or 'hard' opponents.

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.**

    Please see file Assignment4CRC.png in the same directory as this document.

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

    Iterators are likely needed for this problem as the board state has to be analyzed for a victory condition. Storing the board state in some sort of iterable object, and iterating over it and checking for some condition(s) should provide a simple mechanic in checking the board state for a victory.

    For example, many game boards are represented as mathematical graphs (directed or undirected) with vertices and edges. Graph traversal to find unconnected vertices or chains of vertices is one way that we could determine if four tokens have been arranged in a line.

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

    Separating core application logic from GUI logic is the first defence. By directly separating the two the GUI code should become more trivial, thus, easier to test.

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.

There are many programs available for designers that can simulate the colour spaces of every kind of colour deficiency. We will use one of these tools as part of our manual testing to ensure that the game components are recognizable regardless of vision deficiencies.

References

Project source code repository:
https://github.com/ECE421/disconnect-4-hahaha

Chromedriver webdriver:
http://chromedriver.chromium.org/

geckodriver W3C webdriver:
https://github.com/mozilla/geckodriver

Selenium automated web browser engine:
https://www.seleniumhq.org/

Android Studio:
https://developer.android.com/studio

Android UI testing with Espresso:
https://developer.android.com/training/testing/espresso

IOS UI testing with EarlGrey:
https://github.com/google/EarlGrey