ECE 421 Project 3: Connect 4 and Toot-Otto in Rust

Team 1: Prabh Kooner, Jiannan Lu, Brandon Hoynick
Repository: https://github.com/kooner27/421_projects/tree/main
April 12, 2024

Major Innovations

To help handle color vision deficiency, we made the webpage in consideration of if the player had only grayscale vision; this included things like black font and various shaped player pieces that don't rely on color to be noticeable.

We also enhanced the user interface of the board with a move predictor that indicates where the places the tile will land when hovering on the columns.

Design Questions

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

There are several ways a computer can augment the physical game experience. Computers can simulate opponents with varying levels of difficulty so you can improve your skills, even when a human opponent is not available. Computers can provide various skins quickly for the playing pieces (while a printed board requires a mass physical selection of a set). Computers can also provide accessibility features, additional setting adjustments, online multiplayer, game history, statistical analysis for an overall more robust and dynamic gaming experience.

2) What is a computerized opponent? What are its objectives? (Remember, not everyone is an expert.) - What characteristics should it possess? Do we need an opponent or opponents?

A computerized opponent is a software program designed to play games against humans. The objective of a computerized opponent is to provide a challenging and engaging experience simulating the decision and actions of a human opponent. Ideally, the computerized opponent should be able to adapt to the player's skill level providing a good match for both beginner and advanced players. Another characteristic of a computerized opponent should be consistency, unlike humans, who can vary performances depending on external factors, a computer should provide a consistent level of difficulty.Computerized opponents should also be fair and follow the rules of the game.

We should have multiple computerized opponents in order to meet our objective of providing an engaging experience for all skill levels. In regards to our Connect 4 and TOOT-OTTO game we should have easy and hard modes.

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

Interface components especially for a full stack Connect 4 and TOOT-OTTO game involve several considerations. The color scheme should provide good contrast between elements such as the game board, tokens and the background. We should consider color blindness accessibility by choosing tokens that are distinguishable not just by color. Dimensions, and other user interfaces should ideally be rescable and adjust to different screen sizes and orientations. Scroll bars should be used for navigating content exceeding the window sizes.

We did use Google's browser Inspection tool to check for color deficiency (as outlined here: https://learn.microsoft.com/en-us/microsoft-edge/devtools-guide-chromium/accessibility/test-color-blindness) and found our website was:

- pretty good with the different color deficiencies, as the text, icons, and grid were very usable,
- ok with blurriness, as only the small text items were hard to read (but we considered this a extreme blurriness setting),
- not so good with low-contrast, as this makes the game grid near-impossible to see, but, regardless, still playable.

4) What does exception handling mean in a GUI system?

In GUI systems exception handling refers to catching and responding to runtime errors arising from user interaction. In command line interfaces it can be straightforward to handle errors and sanitize user input. In a GUI system, this must be reflected visually. For example, in regards to our connect4 web application, we would need to handle user input errors graphically. For instance, attempting to place a disc in a column that is already full should not be allowed. Recovery mechanisms should also be provided like instructing the user to refresh the page if something goes wrong or does not load correctly.

Ultimately, GUI exception handling can take many forms transcending traditional program exception handling such as having backup source links for fonts and images.

In regards to our Connect 4 and TOOT-OTTO game the most important GUI exception handling is ensuring consistent state maintenance. We need to make sure the state of the graphical board matches the state of our backend rust struct board.

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

A command line interface will allow us to separate the UI logic from the game logic. This makes debugging much easier. It enables us as developers to test core game functionality like wind condition checks and player moves without the overhead of a graphical interface. A command line version of the game is more portable and accessible and can be used on more machines

especially since we are using rust to generate a binary. Finally, a command line interface can easily be extended to a separate REST API backend.

Limitations

The only gameplay bug we encountered is that it is if after playing a winning move the computer also plays a winning move. This is strictly in the frontend as it seems to only check the state after both moves.

Our computerized opponents can be improved. Currently, the easy computer opponents make random moves. The hard opponents consider the column location of the last move and make their move into that column or an adjacent column with a probability distribution of 30%, 40%, 30% respectively. We could explore other techniques to make more difficult opponents by applying minimax and alpha-beta pruning techniques.

Another limitation of our application is that the human player is always Toot. Toot always goes first in the actual game but we could allow the player to choose Otto and have the computer make the first move, but our computer moves were reactive moments based on on_click events that the player had to do; we could have the roles switched, but it would require the player to trigger the computers move (and the computer would select its own spot, this was the case for both Connect-Four and Toot-Otto games), but we felt this option was not as clean as the previous setup we went with where Player1 would always go first.

In the official rules of TOOT-OTTO each player takes six O's and six T's; however, in our implementation they get unlimited pieces and play until there is a winner or the board is full which means there is a draw. It is also worth noting that if we were to extend our game to incorporate various board sizes it would not be feasible for each player to only have six pieces.

User Manual

1) Assuming you have rustc and cargo installed. Install trunk. Trunk is a wasm bundler.
    $ cargo install trunk

2) We also need to add the wasm build target.
    $ rustup target add wasm32-unknown-unknown

3) cd into the yew-app folder. And run the trunk serve command. This will build the application and start the local development server
    $ trunk serve
4) Go to http://127.0.0.1:8080/ in your browser and enjoy!

The command line version is also available as a cargo project located in the “backend” directory. Simply, navigate there and type “cargo run”.

References

https://yew.rs/
https://learn.microsoft.com/en-us/microsoft-edge/devtools-guide-chromium/accessibility/test-color-blindness
Font used: https://www.fontspace.com/category/video-games
Game Icons used: https://opengameart.org/content/fantasy-icon-pack-by-ravenmore-0