* Title of the project: **Logiczzle**
* Project collaborators: **Gabby Wright, Michaela Ann Murray, Yifei He**
* Demo link:[**http://34.201.77.245/logiczzle/**](http://34.201.77.245/logiczzle/)
	+ Note that the HTML code generated by PHP can be directly plugged into Worksheets and will work as expected. In the program, PHP is only used to generate the table and organize the code. The core functionality is unrelated to PHP.
* Source code: <http://34.201.77.245/logiczzle/php.zip>
* **What is the problem? It is important to clarify what the assumptions, inputs and outputs are.**
	+ The problem we tackled was the challenge of implementing a logic puzzle game which generated and solved logic puzzles similar to the Zebra problem we saw earlier in the quarter.
	+ In the puzzle generator, the inputs are a dimension m×n. The output is a puzzle of size m×n (m rows (categories) and n columns (items)) and a sufficient number of clues to allow the puzzle to be solved.
	+ In the puzzle solver, the input is any number of clues (in a format of “<EMOJI> <verb> <EMOJI>”). The output is a solution, if there is one, to the puzzle.
* **What is your solution? What worked? What didn't work? If you made any optimizations make sure to highlight them. If something didn't work well, make sure to highlight it too.**
	+ We used a combination of emojis and predetermined verbs to create a fun and engaging user interface across all different areas of the game. The two main components are as follows:
1. Users can solve randomly generated puzzles created by the app. After specifying the dimensions of a grid, a randomly generated puzzle with those dimensions will be displayed to the user with sufficient clues to solve the grid. As the user selects items for a given cell of the grid, the background of the cell will change to red if their selection directly conflicts with a clue, indicating that there is an error. If all cells have been assigned to an emoji and there are no red cells, the user has successfully solved the puzzle.
2. Users can also have the app solve a puzzle for them. In this part of the game, the user specifies the size of the game they want solved. Then, the user defines a series of rules based on a list of specific criteria and uses the previously defined format of “EMOJI verb EMOJI”. The verb can be any of the most common words in the English language; however, the solver will narrow down the specified verb to a list of six predetermined verbs: is, is\_right\_of, is\_to\_left, is\_next\_to, is\_at\_location. From there, the solver will output the final solved game in a table, much like the table the user fills out in the above randomly generated puzzle.
	* One optimization in terms of solving puzzle part is that we would reorder the clues and first satisfies less expensive constraints (clues) related to `is\_at\_certain\_location`, then `is`, and finally `is\_to\_left\_of`/`is\_to\_right\_of`/`is\_to\_either\_side\_of`. This significantly increases our solver’s speed.
	* Another optimization is that when we are generating the clues, we decided to wait until at least every item in the solution is in a clue to try to test if current clues are enough to solve the puzzle. We did that because we know that if there are still items that are not in any clue, the puzzle cannot be solvable. This makes our generation process much faster.
	* Challenge: Integrating logic programming with a user interface that depends on HTML elements and CSS styling was a challenge. The deductive nature of the programming language sometimes made it hard to link up epilog with the HTML. For example, we decided to use PHP to generate the table (although data is still filled by logic programming) because Epilog does not provide a way to interact more complicatedly with HTML element (e.g., unlike jQuery).
* **If you were faced with multiple alternatives and you decided to choose one among them, justification for your choices e.g. why did you choose a particular data model, rules etc.**
	+ We chose to use emojis because they reasonably limited the scope of the types of inputs and outputs we needed to process. Furthermore, the language that can successfully be communicated with emojis can transcend other language barriers, and emojis are cute and fun.
	+ At first, we have tried to generate clues by elimination. In other words, after we generate all possible clues, we delete rules randomly one by one. Later, we realized we could simply do the reverse, which is randomly choosing clues bottom-up until the puzzle is solvable. This greatly decreased the time needed to generate the puzzle.
* **Is Logic Programming critical/good to have/not necessary for your solution?**
	+ Logic programming is necessary for the majority of our solution since our end goal is a logic puzzle that is best solved using a logic program. Logic programming greatly helped us to solve and generate the puzzle due to properties such as its two-way relation. There are some areas of our solution, such as the text parsing and the handling of HTML elements, that do not necessarily need logic programming in order to be implemented effectively. However, coding both functionalities using Epilog allows for better communication between the other logic programming parts of the project and continuity of the coding paradigm in use.
	+ The table generation for solving the game uses PHP (as mentioned above). PHP is also used to `require` (i.e. include) the rule so that we can decompose ruleset in different files.
	+ Although many elements of the UI design would somewhat more difficult to implement in a primarily logic programming context, the style rule predicates were much cleaner to implement using epilog. Based on a defined conflict predicate, cells would seemingly automatically be colored red if they satisfied the conflict. The code for this was easier to write than it would have been with straight JavaScript. Same goes for `value` and `innerhtml`.
* **How would you extend your work – for projects with worksheets it could be something along the lines of scaling the worksheet for solve a generalization/expansion of the problem?**
	+ Currently the solver is half-finished, with no nice UI to display the answer. That could definitely be improved by showing the user results in a table.
	+ Clues and variables that are all natural language rather than emojis and simple phrases.
	+ Allowing for more complex clues, such as “X does **not** belong to Y” or “X is somewhere to the left of Y (not necessarily next to)”.
	+ Allowing users to take more expressive notes (mark in a cell what they know is not true).
	+ Indicating conflict that is more complex than any single rule alone indicates.
		- So if two rules together indicate that there is a conflict in the selection that the user has made, the cells will be made red based on that as well.
	+ Adding in error checking for incorrect inputs would be a much-needed enhancement, especially if we intend to gear the game towards a larger audience.
	+ Using statistics to check e.g., time used, user aggregate data to analysis the difficulty of each generated puzzle.
	+ Add a difficulty setting that limits the frequency of hints such as `is\_at\_certain\_location` to increase difficulty.