Meghan King
CAP4621
Due Monday, October 27, 2014
Initial Report

0. Insight into my current implementation

To give more insight to my implementation of the game, I will describe the different representations that are currently implemented before delving into the intelligence sections that are required. My implementation is done in LISP, which changes the approach normally taken.

Each line in the game can be in a few different states:

- line - it is a line on the board,
- x - this can never be a line,
- empty - it has not been determined if a line can be drawn there or not, and
- xx - this is a line that is off the board but is used in the validity checking.

There are also multiple map representations that have been implemented to aid the intelligence of this game.

- linerepmap - a list containing the list of each row, with each row containing the lines in that row.
    - the first row is consists of all xx's representing the "top" line of each of the intersections in the top row.
    - the second row contains the left and right lines in the intersections in the top row, also seen as the top part of each tile
    - the third row contains the bottom line of each of the intersections in the top row, or the left and right lines of each tile
    - etc…
- dotrepmap - a list the list of each row, with each row containing all the interactions in that row, which each dot has reference to the top, right, bottom, and left line in that intersection.
    - references between these lines, and the linerepmap are all shared, allowing for updates in a line's state to be shared across all maps.
- tilerepmap - a list containing the list of each row, which contains the number restriction on that tile and references to the lines of the top, right, bottom, and left lines of each tile in that row
    - again, references of lines are shared

The intersections in Slither can be seen in the online version as a dot, and in our version as a '+'. The dotrepmap and tilerepmap I have constructed allow me to utilize the validity checks of tiles and intersections to be done using the mapcar function.

Meghan King
CAP4621
Due Monday, October 27, 2014
Initial Report

1. Intelligence you intend to implement

The intelligence I definitely want to implement is when there are clear constraints on each square, limiting the representations of each square utilizing those constraints. These include, restrictions on intersections and intersection on tiles.

Each intersection can only hold these representations by the end of a correctly completed game:

- 2 lines and 2 x's
- 4 x's

I already have a method that checks the validity of each dot. The validity is checked by checking if their state matches the state of either representation above. This validity check can always be changed to allow for the representations to be altered as well. There are many states that can occur that are not correct, but can be altered into a correct solution automatically.

- 3 x's
  - the other part has to be an x
    if not, a line would mean dangling end
- 1 line and 2 x's
  - the other part of the intersection has to be a line
  - if not, another x would mean a dangling end
- 2 lines and 0 or 1 x's
  - two other parts must be x's
  - if not another line would mean there is a loop in the circuit

There are also states for intersections that are not invalid, but are ambiguous of their final state. These intersections cannot be automatically manipulated to a correct and final state.

- 2x's and no lines
- 0 to 1 x's and 0 to 1 lines

There are also representations of the intersections that are my current states. I will list them here, and will check them in the validity of an intersection, but will discuss this more in the second section of my implementation.

- 3 lines and 0 or 1 x's
  - a loop has occurred
- 4 lines
  - a loop has occurred
- 1 line and 3 x's
  - a dangling end has occurred

The number restrictions in each tile can be used alongside the intersection restrictions to gain more insight on the line placement

Each tile is limited in their representations as well, and are intuitive, such as ruling out all walls of a 0 tile with an **x** state. There are however, many representations of the lines surrounding tiles that will force a tile to be a certain representation, depending on the tile restriction. Many of these restrictions can be accounted for when checking in conjunction with the dots.

For example, say you have a 3 tile with all empty walls. If an intersection on it's corners has it's other two lines with x's, the two lines that are walls of the three tile *must* be in a **line** state. If one of those walls was an x, the other must be as well (due to the intersection restrictions). These tile restrictions will allow us to utilize both their rules and the intersection rules together to find the correct solution of the board.

Utilizing the edges of the board is also a useful strategy, which is why I add the outer edge's line representation in the beginning of the game. The logic behind the edges are the same as any other lines that are of the **x** state, which allows functions to not have to worry about such edge cases.

Checking if their is a correct closed loop is already implemented in my current game. However, it is limited in functionality, since it can only check that at the end of the game, you have implemented a closed loop. This is because the function will find the first line it sees, and checks if that is a closed loop. I plan to add implementation that given a place it could potentially put a line, will you cause a closed loop not containing all the other lines that are currently drawn on the board. I also plan to use that in conjunction with the validity I have of each intersection and each tile on the board to find if the board is in a valid end game state.

Meghan King
CAP4621
Due Monday, October 27, 2014
Initial Report

2. This is what I would love to implement provided I don't have any major difficulties.

To start off with this section, many of the major difficulties I have faced in the past, and will continue to face, is due to my lack of knowledge and experience with LISP. There is a learning curve with this language that has a tendency to cause obstacles in my implementation process. I am continuing to learn the language, but am realistic that it will continue to be my biggest challenge.

I want to revamp many of my maps to be able to use a 'mapcar' function on one list, including the dots and tile representations on my map. I am hoping this can be accomplished. This may not be done due to the dependencies my map representations have on the validity checking of the game. As seen before, my functions with tiles and dots are "lists of rows of …" which causes for extra steps to be taken when wanting to utilize a mapcar function. I am hoping to just have a "list of all dots" and "list of all tiles" in the dotrepmap and tilerepmap, respectively, which will allow me to utilize this powerful function in the LISP language.

I would love to implement the search tree implementation of solving a board when there are no *clear* restrictions on the tiles or intersections, as discussed before. I think this may be hard to visualize in my head due to my lack of knowledge of the LISP language. The main reasons I forsee this being a large challenge since it will be a large amount of memory, or time per move.

I am not aware of how to deep-copy lists to share their new values, just as I do in constructing my first map in the beginning of the game. All of my map representations share references of their lines in memory. Without this, the many validity checks would become useless. I would then need to change how I solve problems by deriving each dot and tile representation for each validity check, which I feel would take up to much time per move. If not, I would need to update each map manually with a change in state, which could be possibly, but not as time or memory efficient as sharing references can be.

These difficulties will definitely shape the AI part of this implementation, and may leave my program to being insufficient against the harder, larger boards that will be given to us during the tournament. This allows me to have a great learning experience with learning that modeling real world intelligence can be tricky.