**HOW TO RUN**

-Load mexicantrain.lsp into LISP interpreter and then execute function (mexicanTrain)

-Menus are operated almost entirely numerically except for filenames to load and save serialization files. Simply enter the appropriate number corresponding to the menu choice you wish to select in playing the game.

**BUG REPORT**

PRINT-PRETTY

**FEATURE REPORT**

Serialization/deserialization

Main Menu

Start of new game

Start of new Round

Computer/Human drawing 16 tiles from boneyard

Correctly displaying all round information

**DATA STRUCTURES/MODULES**

-The client module contains only client code to load additional files and start the game

-The game module manages new games, resuming games, and victory when the player is done

-The round module manages new round, current rounds, ending rounds, alternating turns, and determining victory of round

-The utilities module contains a variety of helpful bottom level functions to aid in design by providing all the tools to design in a purely abstract functional way. Functions include list shuffling/reversal, getNth, setNth, remove, element checking if a list contains element, getting last element, and validating console input

-The boneyard module maintains the boneyard and handles drawing of tiles

-The human/computer/player/help module handles the players turns and Help

**DAILY LOG**

**09/28/21**

-Created game client and util modules, established a way to get valid numbers from client input, established intro menu, and began working on concept of round.

**09/29/21**

-Created round module to begin starting of round. Am passing around all 10 major elements of the game list around to each function. Want to find a way to create getters/setters on this as soon as possible, so I can just pass around the entire game between recursive calls of things like (playRound game) instead of (playRound roundNumber computerScore computerHand…and so on.

-Create serialize and deserialize modules, populate with get and sets for all 10 dimensions.

They are implemented with some very long combinatins of first and rest but I would like a better way. I would like a getNth and setNth that can work on any list.

-I experiment with formatting. The LISP IDE I use formats stuff weird so its best to output lists with ~d in the end.

-Setup recursive startRound to handle, one by one, the establishment of boneyard (if its null) (still not shuffled), the drawing of all tiles to be drawn, and the flipping of the coin (if nextPlayer is null)

-getNth, setNth, and many more utility functions added. At this point I am pretty sure I have every single util function I need to perform all the game actions, with the exception of a recursive add all. One that can add together the 2 pipcounts in the list that is each tile, while summing up all the times. This is the job for at least 2 functions, with one calling the other.

**09/30**

-I have completed a game loop and a playRound loop. playRound is not tail recursive, but endRound returns list of both players score, which are returned back through every round, back to playGame which uses those scores to start fresh new round with 1 higher round number. This way I am going much easier on the stack, only 1 level deeper per round of game played.

-I have tacked on an 11th field to the game list that gets passed around. This field is a list of predicates representing whether each player had passed in last turn. I anticipate needing this as I cannot conceive of any functional way to calculate this. It is one of the few state variables that must be created and maintained throughout a round. I believe that if my plan for player modules goes as planned, then I will need no other state variables.

-I have created player class and given it the function tallyHand and the skeleton of one called endturn. A players turn will be initiated from round, and the rest of it will be handled in the respective player classes, but the recommendation system must come from player class and be available to both.

**10/01/21**

-I have been forced to refactor everything now that I understand functional programming a bit better. I have setup some very long chains of returns that kick all the way back to playRound. This final return is the new game state after all modifications have been complete. Another innovation was to return more than just the game state from the function that plays tiles. Now, by returning a list of the game state, and some informationa bout the play, the necessary steps can be taken based on that info, and can be done in a more appropriate part of the program.

I am rapidly creating many empty functions, aiming downward, only occasionally stopping to implement any terminal nodes that shoot off to the side. The great thing about LISP is that I don’t even have to change files to test a function. I can make all sorts of test calls just underneath of it in order to test it. The convenience of that and power is truly exceptional.

**01/30/21**

-I have established the major of what it takes for a player to make a move, but with many stubs along the way. Markers aren’t correctly handled, no passing recording is occurring. I am finding, however, that in order to add these things is usually no more than 1-2 levels of function scope from each other, and just as Professor said, it does not impact any other part of the program.

**10/02/21**

-I have implemented canPlayTileToTrain and playTileToTrain, as well as finalizePlayTile.

remove marker in playTileToTrain if trainnumber = player supplied train number that needs to be added as argument to that function.

add ability to kick back to inputting train when you are inputting tile, or try inputting them in pairs and validating at the same time.

remove the superfluos train number argument from human moves and the return list from finalize move

**02-04-21**

-help.lsp, player.lsp, human.lsp, computer.lsp have all been populated. The framework for the entire rest of the game has been laid out, but some refactoring of files may be needed.

-Most of these functions are empty, but their flow has been tested with stubs and it works. Now I can officially say the top down design is done and the rest is a matter of implementation and testing individual functions, replacing stubs with logic one by one.

**02-06-21**

-4 hours logged working. Updates made to help and human. Implemented getBestDouble and tested, as well as getBestSingle.

**02-07-21**

-2 hours logged working. Updates continued on help and human. Implemented getMoveReason, getBestMove, checkOwnTrainPriority

**02-08-21**

-A few more hours logged. The last of help.lsp, checkPlayAgainstMarker, getBestTrainForTile, and I ended up adding a canPlayHandToTrain (for convenience, a refactor would have prevented the need for this, but didn’t seem worth the effort.

**02-09-21**

-Computer.lsp is created as a partial copy of human.lsp. This creates some code smell, but without objects it is not as simple to solve. I feel having modular player-universal functions would end up being just as sloppy without polymorphism to aid in the process. I would constantly be doing conds on playerNumber, which isn’t much better to me.

**02-10-21**

-The computers turn is hooked up to the help system (with a few modifications to differentiate it, mostly cosmetic changes to the code). I spent a few hours refactoring and have begun adding more comments and function headers.

-Later that night: Have completed the game as far as the lisp files go. I will continue to test extensively all 4 cases.

**02-11-21**

-Am completing manual and obtaining screenshots. Gearing up for submission.

**SCREENSHOTS**

**Text

Description automatically generated**

**Starting new Game, coin toss, round setup.**

**Text

Description automatically generated**

**Resuming/Saving game state**

**Text

Description automatically generated**

**Input Validation on all menus, including filenames**

**Text

Description automatically generated**

**Drawing from boneyard, passing, Help, Making moves, going back a menu, tile input, train input**

**Text

Description automatically generated**

**Playing doubles and getting to go again**

**Text

Description automatically generated**

**Round end, prompt for new round. Start new round.**

**Text

Description automatically generated**

**Announce Winner**