**Rubric for Assessing Mexican Train in LISP**

Name: Michael Whitlock\_\_\_\_\_\_\_\_\_

Email: [\_\_\_\_mwhitlo1@ramapo.edu](mailto:____mwhitlo1@ramapo.edu) \_\_

Carefully **highlight** **all** the items that **work correctly**.

Incorrect entries may be penalized. Not all the entries may be used for grading.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Setup of the Game** | | | | | |
| **Players** | One player is Human |  | One player is computer |  | Players alternate |
| **Double-9 set** | Contains 55 tiles |  | Tiles are unique |  | Tiles are shuffled before each round |
| **Setup** | Engine is placed on the Table |  | Engine is 9-9 for the first round | Engine changes on each round |  |
|  | 16 tiles are dealt to each player |  | 22 tiles left in boneyard |  | Computer, human and Mexican trains start empty |
|  | Computer’s train is to the left of the engine |  | Human train is to the right of the engine |  | Mexican train must be started with a tile matching the engine |
| **First player** | Player with lowest game score plays first |  | If tie in score, human is asked to call the toss | If human calls correctly, human plays first | Otherwise, computer plays first |
| **Human Player** | | | | | |
| **Turn** | Plays at the end of human train | Plays at the end of computer train if marked | Plays at the end of Mexican train |  | If an orphan double on the Table, plays only against the orphan double |
|  | Draws a tile from boneyard if no playable tiles in hand | Passes the turn if drawn tile is not playable | Adds non-playable tile to the hand | Puts marker at the end of human train when passing turn | Removes marker only after playing a tile on human train. |
|  | Can play double and non-double in the same turn | Draws a tile from the boneyard if second playable tile not in hand |  | Can create an orphan double by playing the two tiles on separate trains |  |
|  | Can play two doubles and a non-double in the same turn | Will do so without drawing a tile from the boneyard |  | Can create one or two orphan doubles |  |
| **Computer Player** | | | | | |
| **Turn** | Plays at the end of computer train | Plays at the end of human train if marked | Plays at the end of Mexican train |  | If an orphan double on the Table, plays only against the orphan double |
|  | Draws a tile from boneyard if no playable tiles in hand | Passes the turn if drawn tile is not playable | Adds non-playable tile to the hand | Puts marker at the end of computer train when passing turn | Removes marker only after playing a tile on computer train. |
|  | Can play double and non-double in the same turn | Draws a tile from the boneyard if second playable tile not in hand |  | Can create an orphan double by playing the two tiles on separate trains |  |
|  | Can play two doubles and a non-double in the same turn | Will do so without drawing a tile from the boneyard | Can create one or two orphan doubles |  | Prints rationale for its move |
| **Ending/Scoring** | | | | | |
|  | Round ends when either player plays the last tile | Round ends when boneyard is empty and both players pass |  | Round scores of both players printed and correct | Game scores of both players printed and correct |
|  | Player asked if another round should be played | If yes, another round is started | If no, winner of the game is printed | Winner is the player with the lower game score | Game can be a draw |
| **Implementation Features** | | | | | |
| **Serialization** | Provides option to save game before each player’s turn | Game is saved into text file | Correct format used for text file | Game state correctly saved | Game quits upon serialization |
|  | Provides option to resume game from text file | Prompts for the name of the text file | Resumes game from saved state |  |  |
| **Correctly**  **Restores** | Round number |  | Next player |  |  |
|  | Computer’s score | Computer’s hand | Computer’s train |  | Mexican train |
|  | Human score | Human hand | Human train |  | Boneyard |
| **Help mode** | Has the option to ask computer for recommended move |  | Computer uses its own strategy to recommend the “best” move |  | Prints the rationale for its recommendation |
|  | Recommends which tile(s) to play from hand |  | Recommends which train(s) to place the tile(s) |  |  |
| **Game Features** | | | | | |
| **Input Validation** | Asking whether to play another round |  |  | Saving a game to a file | Resuming a game from a file |
|  | Asking for help |  | Which tile(s) to play from hand | Which train(s) to place the tile(s) |  |
| **Output**  **Clarity** | Both scores correctly updated | Both hands correctly updated | Both trains correctly updated | Mexican train correctly updated | Boneyard properly updated |
|  | Round number displayed | Next player clearly identified | Round points displayed | Game points displayed | Game winner clearly announced |
| **Design** | | | | | |
| **Functional design** | Functions defined for any code executed more than once | Each function in charge of only one logical task | Each function is self-contained | Overlap between functions is minimal |  |
| **Code Design – Data flow** | Data flow is through return values and parameters only | Data propagated through function composition | Appropriate list structures used to hold data | Appropriate accessors and manipulators provided for each data structure | Changes to data always validated |
| **Code Design – Control** | Overall design is hierarchical, top-down decomposed | Function call chart is a balanced tree, not a tail-recursive chain | Overall structure evident in each top-level function | No more than 3 levels of function calls for a task (excluding recursion) |  |
|  | Code for repeated execution separated from  code for single execution (e.g., of round, game) | |  | Display concerns separated from problem logic (Model Vs View) | |
| **Implementation** | | | | | |
| **Identifiers** | All variables have names corresponding to nouns in problem description | All functions have names corresponding to verbs in problem description | Any abbreviations in the names are readable |  |  |
| **Coding style** | No global variables used | No set functions used | No imperative constructs used (if, while, for, prog, arrays) | No destructive constructs used | All literal constants are explained at each occurrence |
| **Courtesy Programming** | | | | | |
| **Listing** | Code is indented properly | Code does not wrap around to the next line | Functions listed in the order in which they are called | Functions are grouped according to their functionality |  |
| **Documentation** | Every function has a complete header | Within each function, code is properly commented – steps in the algorithm are listed | Comments in the code describe semantics, not syntax | Comments in the code do not have spelling/ grammatical errors. |  |
| **Submission** | | | | | |
| **Manual** | Describes how to run the program |  | Includes screen shots for each user input | Includes screen shot of game setup (hands, trains, boneyard) | Includes screen shot of computer providing help |
|  | Includes bug report |  | Includes missing features report | Includes additional features report |  |
|  |  |  | Includes project log | Source and documentation are placed in a directory and the directory is zipped | |
| **Milestones uploaded?** | First:  Yes / No | |  | Second:  Yes / No | |

**Please describe the strategy used by the computer:**

* **Which tile(s) to play from hand**

**First, doubles are checked in descending order. The highest possible double is played. If none can be played, The next check is against players own marker. If player can make a move to remove their own marker, they will play the highest pip count tile that can remove marker. Otherwise, they play the highest pip count tile that can be played anywhere.**

* **Which train(s) to place the tile(s)**

**If the players own train has a marker, this is the priority unless theres a double of higher value that can be played elsewhere.**

**The player will attempt to play on its own train next, followed by opponent, followed by mexican**

**Do not delete these last three pages.**

Input validated

Help explained

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.1 Help:

Move:

1.2 Move:

Explanation:

1.3 Help:

Move:

1.4 Move:

Explanation:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.1 Move:

Explanation:

2.2 Help:

Move:

2.3 Move:

Explanation:

2.4 Help:

Move:

2.5 Move:

Explanation:

2.6 Help:

Move:

2.7 Move:

Explanation:

2.8 Help:

Move:

2.9 Move:

Explanation:

2.10 Help:

Move:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.1 Help:

Move:

3.2 Move:

Explanation:

3.3 Help:

Move: