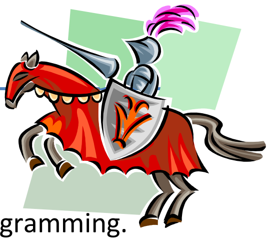
CS 4700
Scheme Programming Assignment

(Jousting (with (Scheme)))



Introduction

This assignment is aimed at providing an introduction to using Scheme as well as functional programming. Because of this students will be held to only using the functional constructs of the Scheme language – that specifically means no iteration and no variables!

This assignment pits virtual knights against one another in epic jousting tournaments until only one knight remains.

Rules

The object of the game is to unhorse your opponent, without you being unhorsed. In each joust a knight will have a *shield position* (high, low, or duck) and a *lance position* (high or low). A knight unhorses their opponent if:

1. Their lance is high and their opponent's shield is low
2. Their lance is low and their opponent's shield is high
3. Their lance is low and their opponent's shield is duck

All other combinations result in their opponent remaining on his horse. You are unhorsed if your opponent unhorses you. During one joust it may be the case that

1. Both knights are unhorsed, in which case the game continues
2. Neither of the knights are unhorsed, in which case the game continues
3. The first knight is unhorsed, but his opponent is not, in which case the opponent wins
4. The opponent is unhorsed but the first knight is not, in which case the first knight wins

Instructions

Setup Part 1: Download the joustFiles.zip archive from canvas. The archive contains four files, knights.scm, game.scm, tournament.scm, and test.scm. You will fill in the function stubs in these files as described below and test your code with test.scm.

Setup Part 2: Download and install a Scheme interpreter. I would suggest using Chicken Scheme (Windows: https://bitbucket.org/roti/chicken-installer/downloads, others: http://wiki.call-cc.org/platforms) or Dr. Racket (http://download.racket-lang.org/). Both should work in Windows, Linux, or Macs. If you use Dr. Racket, make sure to set it to Scheme (in the menu goto Language -> Choose Language…-> Other Languages ->R5RS).

Each knight is defined by a Scheme function that **when evaluated returns a list** consisting of three items: (*shieldPosition lancePosition name*). Four example knights: sir-robin, black-knight, king-arthur, and joan-of-arc, are defined in the provided knights.scm file. Some example calls and their results would be:

Call	Result
(black-knight)	(duck low "Black Knight")
(black-knight)	(low low "Black Knight")

Step 1 (20%): Define a function, *joust*, inside of game.scm which takes two knights as parameters.
(joust knight1 knight2)
The result of this function is a list containing two Booleans corresponding to **which knights are still on their horses**. If both knights are unhorsed the result would be *(#f #f).* If the first knight remained on his horse while his opponent was unhorsed the result would be *(#t #f).* Add some output to this function so we can see what is happening.

Step 2 (20%): Define a function, *jousting-game*, inside of game.scm which takes two knights as parameters.
(jousting-game knight1 knight2)
This function will repeatedly call *joust* until a winner is decided. If the first knight parameter wins, *1* is returned. If the second knight parameter wins, 2 is returned. Don't forget to add some output to this function so we can see what is happening. You may want to include some condition here to avoid a deadlock condition if two knights do the same thing every time which always triggers another joust although this is not required.

Step 3 (10%): Define two new knights ***(sir-lancelot)*** and ***(yourFirstName-yourLastName)*** inside of knights.scm. Lancelot has a 0.4 probability of holding his shield high, a 0.4 probability of holding his or low, and a 0.2 probability of ducking. He also has a probability of 0.5 for each lance position.
Next create your own knight with what you think to be a winning strategy. I will place all of the class's knights against each other in a tournament. The knights who perform best in these tournaments will receive extra credit on the assignment!

Step 4 (30%): Define a function named *tournament* inside of tournament.scm which takes as its only parameter a list of knights participating in the tournament.
(tournament knightsList)
The result of this function is the index (1-indexed) into the input list of the victorious knight. For example: *(tournament (list sir-lancelot joan-of-arc black-knight sir-robin))* would return *2* if joan-of-arc won.
The tournament is to be a double-elimination style. For the non-sports fans out there, details on how to run a double elimination tournament can be found here http://en.wikipedia.org/wiki/Double-elimination_tournament or here http://www.printyourbrackets.com/double-elimination-tournament-brackets.html. It doesn't need to be anything fancy, just a simple double elimination tournament.
Add sufficient output to show the tournament results.

Step 5 (20%): Define another function named *season* inside of tournament.scm which takes two parameters, the number of tournaments to run during a season and a list of knights.
(season numTournaments knightsList)
The result of this function should be a parallel list of victory counts corresponding to the input list of knights.
For example: *(season 25 (list sir-lancelot joan-of-arc black-knight yellow-knight))* might return *(10 4 6 5)*

Submission

Place your four .scm files into an archive called scheme-yourName.zip and submit it via canvas. Make sure that you have sufficiently documented your source code before you submit it. Enjoy!