Five Crowns Prolog Manual

Author: Frank Ryan

Submitted: 12/10/2019

Bug Report:

Loading invalid load file, and saving invalid directories.

Feature Report:

    Missing features: Validation of load file

                Validation of saving file

    Extra features: none

Description of data structures:

To run my program, the main predicate to call is the beginGame predicate. The predicate takes no parameters, and is to be called as … beginGame. .This predicate will call the mainScreen predicate, beginning the program.

    Lists are used for most data in my program. The player hand, draw pile, and discard pile are all lists. In some functions, such as goOutTest1 and checkRun/Book, I use multiple lists within one list. The input from the human player are recognized as atoms. The round, computer points, and human points are also recognized as atoms. I never use any destructive functions.

    Primary Function Descriptions:

    checkRunFinal, checkRunMiddle, checkRunThird: These predicates are called to find all runs from a card list. The checkRunThird finds all runs of a certain card for a certain size. The checkRunMiddle function stores the run found in checkRunThird for the card of that size, and then looks for a run of a different size of THAT SAME CARD. The checkRunFinal stores all runs found in checkRunMiddle, and is then recursively called to get the next card of the hand, and pass that to checkRunMiddle to get the runs for that card.

    checkBookFinal, checkBookMiddle, checkBookThird: See checkRunFinal, checkRunMiddle, and checkRunThird description. The predicates are the same, except the predicates are looking for books instead of runs.

    goOutTest1: Used to see if a player can go out. The parameters contain a player’s hand, a list of all his runs and books, and the minimum amount of cards left over from books and runs. Passes the first of the list of books and runs to the goOutTest2 predicate. This predicate will return a list of books and runs from the hand. The first of this returned list will be the leftover cards that are not used in books/runs. A condition will see if the length of the first of this list is less than the minimum length parameter. If it is, goOutTest1 is called again with the rest of booksAndruns when it was first called, the same hand, and the new minimum, which is the returned list of goOutTest2. If the length of the returned list is not shorter, then goOutTest1 is called again with the same hand and minimum, but also with the rest of booksAndruns. When booksAndRuns is empty, the best combination of books and runs with the least amount of leftover cards should be returned.

    goOutTest2: When called from goOutTest1, the parameters it has is the book/run passed, and the hand that does not contain the cards in the passed book/run. This function then creates a new list of runs/books with the leftover hand. It is then called again with the first book/run of this new list, and new hand that removes this first book. The base case for this function is when there can be no books/run created from the hand. It then returns all books/runs that were created, and the leftover cards in the hand, if any.

Log:

Milestone #1

From **11/16 to 11/19 …**

Since I finished my Java/Android project earlier than I expected, I decided to go right into the Prolog project. Even though we have not been lectured about Prolog yet, I went onto Professor Kumar’s website and read his Prolog paper, which described many aspects of Prolog. I quickly noticed that Prolog works very similar to LISP. In the first couple of days, I tried writing simple predicates to understand how they work, as well as how they can be nested and overloaded. After some trouble, I found out the hard way that variables cannot be changed in Prolog, which could have saved me some time in these first couple of days. However, once I was able to understand the functionalities of Prolog, I went onto trying the project. I first created the main screen, and then went onto creating the coin toss predicate. From here, I started designing the layout of rounds and turns. This took me a total of about 7-9 hours.

From **11/19 to 11/24 …**

After contemplating possible predicate nesting and analyzing my LISP project, I found that my LISP project of nested functions can be used similarly to Prolog. So, I created predicates for going through rounds similar to my LISP project, where I have two functions/predicates named startRoundsComputerFirst and startRoundsHumanFirst that deal with the coin toss winner and the round winner. Now that I have the basic round iteration laid out, I decided to go right onto the goOut and checkRun/Book predicates. Again, I looked towards my LISP project to see how to create these predicates. First, because I cannot create simple functions like getValue and getSuite of a card, I decided to translate the hand of cards to numbers+suite (ex. 10c), which is exactly what I did in LISP. However, because each condition must be a predicate in Prolog, I have to create many predicates for this translation, which was time-consuming. Once I created the translation predicates, I could then sort the hand, and then use the same exact predicates/functions of checkRun/checkBook that I used in LISP.

The main issue that I ran into from these translation of LISP to Prolog was separating each condition of LISP into its separate predicate in Prolog. Also, because I never want a predicate to return false, since this would cause Prolog to backtrack, I had to create some minor changes to my conditions in order for at least one predicate to return true. After translating checkRun/checkBook to Prolog, I now look towards the goOut function. I would say this all took me about 10-12 hours in total.

From **11/24 to 11/26 …**

On these days, the whole project seemed to be coming together nicely. The goOut function was easily translated from LISP to Prolog, as there went not as many conditions for this function in LISP, so the amount of predicates used was low as well. Now that the goOut predicate completely works, I basically have my computer and help functionalities as well, since they both rely on this function. Instead of designing the computer, I went ahead and created the help predicates since it would be easier for me to understand any errors that may occur. However, before I went onto this, I had to translate the number cards back to their original values, which was again time-consuming. Once I was finished with this, my help predicates were again very similar to LISP, as the computer in my LISP project was quite effective. After working of these function translations, the computer and ask for help predicates were made fairly quickly, and now, the base game is good to go. I did run into some problems passing who won each round, but realized after some time that I was trying to change a variable that was already instantiated, which was frustrating. From here, I now need to create serialization functionality, and test errors (eg. special cases, invalid input, etc). All of these predicates were created in about 5-7 hours.

Milestone #2

From **11/27 to 11/30 …**

I’m quite surprised that I was able to finish the game rather quickly (excluding serialization). So, this time was dedicated to loaded and saving games. I ran into some difficulties loading a file in, but I was able to find a library call, read-string, that helped me load in all aspects of the game without needing a period at the end of every line from the file. Once, this was done, I went onto saving games into text files. This took me a total of about 3-4 hours.

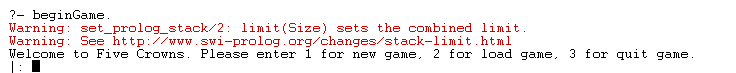
From **12/1 to 12/3 …**

Now that loading game functionalities were done, I easily went onto saving text files, which is just a simple output stream to a text file. After completing this, I now move onto the manual and documentation, since my game is now fully complete. This took my about 1-2 hours.

To run my program, the main predicate to call is the beginGame predicate. The predicate takes no parameters, and is to be called as … beginGame. .This predicate will call the mainScreen predicate, beginning the program.

Screen Shots:

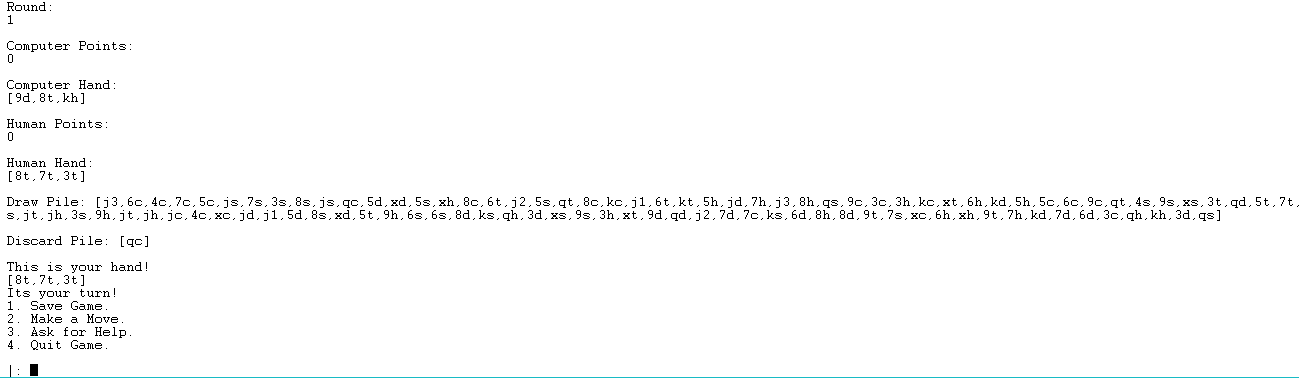
Enter “beginGame.” to bring up the main screen of the game. Here, you can start new game, load game, or exit.



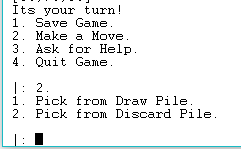
When starting a new game, enter 1 or 0 for the coin toss.



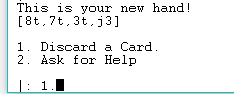
Main round screen shows possible moves of human player. 1 to save, 2 to make move, 3 for help, and 4 to exit.



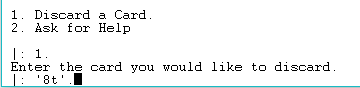
To make a move, enter 2, and then 1 or 2 to pick from draw pile or discard pile.



To discard, enter 1, or 2 to ask for help.



To discard, enter card to discard. If the card is not a face card or joker, put single quotations around the card. If not, put no quotations.



To try to go out, enter 1, or enter 2 to ask for help building runs and books.

