Code Documentation

The application consists of three files, which should be placed in the same directory in order to be runnable. These files are the SlotMachine.cpp, SlotMashine.hpp and the configuration.ini

Implementation Details

1. The configuration.ini file contains the information about the pay-table, reel strips and lines configuration.

   The pay table is represented in the config file as 4 columns starting with the column corresponding to the "2 in a row" column of the pay table in the Atkins Diet paper.

   The next table is the reel strips, as it appears in the Atkins Diet paper, where all the symbols have been replaced by a number from 1 to 11, using the ordering of the pay table (1: Atkins, 2: Steak etc.)

   The third table is the lines configuration, which give information about which row of the 3x5 game_strips table to use each time for evaluation. The values are 0,1 or 2 and there is one value for every one of the 5 columns. This table was created by looking which lines uses the real game in the website for evaluation. The maximum number of lines that can be used for evaluation is 20. This 20x5 table is being read in the lines_config array and according to the current lines_num, the specific amount of its rows is used for the evaluation. Every row of the lines_config array contains one line for evaluation.

   The above three tables are being read into arrays in the readConfigFile method.

2. The initialization with zero of several arrays and variables takes place in the initialize method. This is being done once in the whole running for specific number of evaluation lines.
3. Then a loop until the number of simulation games begins, where for each new game a random game_strips array is produced.

   This array is evaluated in the scoreHitsCalculation method according to the pay rules. This method calls also the scaleCombinations and findCombinations methods which return how many scales are on the screen and how many other symbols occur in a row and how many times accordingly. More details about the way this is being done can be found in the comments of the source code.

4. After the loop reaches its end (the loop for the number of simulation games), the probabilities and the return for each symbol is calculated in the probRetCalculations method and the printResults method in the main prints the results (hits, probabilities, returns, summaries) in a file. The printHitRate is placed outside the loop for testing increasing number of lines evaluation and it prints in a file in the end the hit rate for every different number of tested lines and also the average hit rate from all of them.

For the free spins implementation there is a boolean flag used in the SlotMachine.hpp file, the freegames. Setting it to true calculates also the free spins and setting it to zero it does not.

Results

According to the results, which are placed in .txt files, the return stays always the same independently of the number of lines.

For the free spins the practical bonus gets close to the theoretical and in some rare cases overcomes it. As you can see in the summary results in the output_with_freespins.txt file the theoretical value is 0.266407 (26.6407%) and the practical can be around 0.192476 – 0.282924(19.2476% – 28.2924%).

There are 4 txt files:

1. output_no_freespins.txt (hits, probabilities, returns, summary when running without free spins)
2. output_with_freespins.txt (hits, probabilities, returns, summary when running with free spins)
3. hit_rate_no_freespins .txt
4. hit_rate_with_freespins.txt

Running the simulation for 33,554,432 number of games without free spins and increasingly number of lines up to 20, took 649023msec.

Running the simulation for 33,554,432 number of games plus the free games earned and increasingly number of lines up to 20, took 719671msec.

For improving the performance I tried to use as possible less comparisons, because the time needed is increasing with the number of lines being evaluated. Also I tried to use the memory in a effective way. I believe that there is ground for performance improvement, by for example using instead of arrays in some cases more efficient data structures, maybe maps or lists of vectors, but the time was not enough for testing the performance of such dynamic implementations. Also there is maybe a more efficient algorithm for finding the combinations, which could reduce the number of comparisons made.