Now I want to explore this last idea further with the fictional game Persian Monarchs. As far as I can tell, this game was invented by P. G. Wodehouse for his classic 1939 comedic novel [Uncle Fred in the Springtime](https://www.goodreads.com/book/show/13707721-uncle-fred-in-the-springtime), surely a front runner for some of the best humorous writing of the twentieth century. Wodehouse was in true mid-season form with this, his first full length novel featuring “[Uncle Fred](https://en.wikipedia.org/wiki/Uncle_Fred)” (Frederick Altamont Cornwallis Twistleton, 5th Earl of Ickenham); before much of his work became tired and formulaic in the post-WWII years.

The plot of the novel is complex and not really to the point for a statistics blog, so do yourself a favour and read it for yourself in your own time. But for our purposes, it’s important to know that several of the key plot twists revolve around substantial sums of money gambled on Persian Monarchs, described by one character thus:

‘I could teach it you in a minute. In its essentials it is not unlike Blind Hooky. Here’s the way it goes. You cut a card, if you see what I mean, and the other fellow cuts a card, if you follow me. Then if the card you’ve cut is higher than the card the other fellow has cut, you win. While, conversely, if the card the other fellow’s cut is higher than the card you’ve cut, he wins.’

He shot an anxious glance at Mr Pott, as if wondering if he had been too abstruse. But Mr Pott appeared to have followed him perfectly.

‘I think I see the idea,’ he said. ‘Anyway, I’ll pick it up as I go along.’

Claude Eustace ‘Mustard’ Pott – the “Mr Pott” in the quotation above – is actually a Persian Monarchs player of considerable accomplishment; in addition to being a former silver ring bookie and Shakespearan actor, and current private detective and part time card hustler. So having a not-too-bright but rich and friendly representative of the upper classes explain his own favourite game to him is as close to heaven as he gets.

It’s not clear from the novel whether Mr Pott (or, later, the Duke of Dunstable, who also shows himself to be a master of cardplay) win at Persian Monarchs through superior skill or through outright cheating. There are indications both ways. For my purposes, I’m going to put aside the question of cheating and consider how it’s possible for skill to shine through in such a simple game.

It all comes down to the addition of choices about offering and accepting wagers. As the gambling rules around Persian Monarchs are not described by Wodehouse, I’m going to assume that it’s something like the following:

* each player contributes one counter (or other gambling unit) at the beginning of a hand to a combined stake, before receiving their card;
* after seeing their card, the non-dealer has the option of raising the wager by an extra amount, up to some specified limit (10 in my simulations below), but by zero if they wish. The dealer then has to either cover the increased wager, or decline on the spot in which case the non-dealer wins the existing stake in full regardless of who has the higher cards;
* if the dealer has covered the extra wager, whomever has the highest card wins and collects the entire stake.

I’m also going to assume that once a hand is finished, those cards are put aside and the next hand proceeds with a reduced pack; until the pack is finished at which point the cards are all brought together and reshuffled.

To skip to the chase, here are the results of simulations of three different strategies at Persian Monarchs:

* choosing to offer or accept extra wagers completely at random;
* offering or accepting wagers partly at random (in order to avoid being predictable and easy to bluff) but with probabilities aimed at delivering optimal results in the long run, taking into account which cards have so far appeared;
* as above, but without bothering to count the cards that have already appeared and been discarded, so basing choices on general probabilities about one’s card relative to a complete pack.

At the end of a game that goes through two whole packs of cards, the optimal strategy outperforms the non-card-counting by between 3.8 to 5.3 counters; and the random strategy by 35 to 37 counters. So skill counts big time. Even the marginal advantage given by counting the cards and adjusting one’s strategy accordingly makes a 4% difference in your rate of return.

**‘Optimal’ strategy?**

For what I’ve defined as the optimal strategy (without either proof or adequate demonstration by simulation), the non-dealer’s wager-offering algorithm is as follows:

* First choose whether to offer an increased wager or not. Do this at random, with a ‘probability of offering more’ equal to the estimated probability that the card I’ve got is higher than a random card from the pack.
* Having decided to offer an increased wager, pick a random number between 1 and 10 to increase the wager by that.

The decision of whether to accept or not is taken similarly.

The trick in constructing a decision rule here is that we want to avoid being predictable to our opponent. For example, if we only offered an increased wager when we had one of the top 50% cards in the pack, we have ruled out any bluffing, and once our opponent picks this up they can use it to their advantage. Similarly, if we increase the wager by more depending on our confidence, we are giving away information. So we want a compromise between strict determination and a bit of randomness always present (even if I draw the Queen of spades I might decline an increased wager, or with the 3 of clubs I might offer to increase the stake, albeit at low probability), with your opponent not quite sure where you’re coming from – let’s call it the ‘[madman strategy](https://en.wikipedia.org/wiki/Madman_theory) of Persian Monarchs’.

**Writing a card game in R**

How did I implement all this?

First, I defined a few convenience objects and functions.

* pack is a data frame of 52 rows with the value and suit of all the cards in a standard pack of cards, with precedence in order as per Bridge;
* pm\_draw is a function that, given an existing (possibly incomplete) pack of cards, returns a drawn card and the remaining pack as two elements of a list:

library(tidyverse)

library(scales)

library(parallel)

#' Define a starting full pack of cards. In sequence as per Bridge ie Ace of Spades highest, Two of Clubs lowest.

pack <- data\_frame(

id = 1:52,

suit = rep(c("clubs", "diamonds", "hearts", "spades"), each = 13),

value = rep(c(2:10, "Jack", "Queen", "King", "Ace"), 4)

) %>%

mutate(label = paste(value, "of", suit))

full\_pack <- pack

#' For a given pack (not necessarily a full one), pull out one card, and return

#' that card and the remaining pack (as a list with two elements)

#'

#' @param pack a dataframe with such as that defined by `pack`, with at least

#' a column named `id`

pm\_draw <- function(pack){

drawn <- sample\_n(pack, size = 1)

remaining\_pack <- pack[pack$id != drawn$id, ]

return(list(drawn = drawn, remaining\_pack = remaining\_pack))

}

Then the workhorse of the whole project comes in the next function, pm\_hand, which plays a single hand of Persian Monarchs with two players. The wagering strategy can be “auto” (the best strategy I could think of), “ask” (a human gets asked to decide each wager decision) “random” and “non-card-counter”. In itself, this function is not much use, but it abstracts the work of a single hand of dealing, wagering, accepting and checking the result away from what is going to become the main loop of the game to ge defined later.

#' Play a hand of Persian Monarchs

#'

#' @param p1\_counters number of counters owned by player 1 (non-dealer) at beginning of hand

#' @param p2\_counters number of coutners owned by player 2 (dealer) at beginning of hand

#' @known\_pack pack at the beginning of the hand, 'known' by the players if they have been counting cards.

#' @p1\_method method for player 1 deciding whether to offer an increased wager

#' @p2\_method method for player 2 deciding whether to accept an offered increased wager

#' @verbose Whether or not to print out extra information to console

#' @p1\_lab Label for player 1, used in messages to console

#' @p2\_lab Label for player 2, used in messages to console

#' @max\_wager Maximum additional wager to be offered by player 1

#' @details Player 2 is the dealer and Player 1 gets the first choice - whether or not to offer an increased wager

#' on top of the original wager of 1 counter, and if so how much to make that additional wager. Player 2 then gets

#' to accept and cover that wager, of to fold and concede the current wager of 1 counter to Player 1.

pm\_hand <- function(p1\_counters, p2\_counters, known\_pack,

p1\_method = c("auto", "ask", "random", "non-card-counter"),

p2\_method = c("auto", "ask", "random", "non-card-counter"),

verbose = FALSE,

p1\_lab = "Non-dealer",

p2\_lab = "Dealer",

max\_wager = 10){

# convenience function for printing a message to the console but substituting the correct label

# for p1 or p2 (player 1 or player 2):

pm\_cat <- function(txt){

txt <- gsub("p1", p1\_lab, txt)

txt <- gsub("p2", p2\_lab, txt)

if(verbose){

cat(txt)

}

}

p1\_method <- match.arg(p1\_method)

p2\_method <- match.arg(p2\_method)

# Both players start by putting in one counter each just to start to play

p1\_counters <- p1\_counters - 1

p2\_counters <- p2\_counters - 1

wagered <- 2

# Draw a card each

tmp1 <- pm\_draw(known\_pack)

tmp2 <- pm\_draw(tmp1$remaining\_pack)

p1 <- tmp1$drawn

p2 <- tmp2$drawn

# The probabilities as they appear to the players:

p1\_chance = sum(known\_pack$id < p1$id) / (nrow(known\_pack) - 1)

p2\_chance = sum(known\_pack$id < p2$id) / (nrow(known\_pack) - 1)

# Decision - will p1 wager more?

if(p1\_method %in% c("auto", "random", "non-card-counter")){

# If p1 is being rational, it's chance of increasing the wager comes from it's appreciation

# of the probability of winning this hand (more is 1 if increasing the wager, 0 otherwise).

# But if the method is random, then they choose at random

chance\_more <- case\_when(

p1\_method == "auto" ~ p1\_chance,

p1\_method == "random" ~ runif(1, 0, 1),

p1\_method == "non-card-counter" ~ sum(full\_pack$id < p1$id) / 51

)

more <- rbinom(prob = chance\_more, size = 1, n = 1)

extra <- sample(1:max\_wager, size = 1)

if(more == 0){

pm\_cat("p1 does not add to the initial wager.\n")

}

} else {

pm\_cat(paste0("You drew the ", p1$label, ". How much extra will you bet this hand?\n(maximum is ", max\_wager,

", 0 is acceptable)"))

extra <- -1

while(!extra %in% 0:max\_wager){

extra <- as.numeric(readLines(n = 1))

}

more <- as.integer(extra > 0)

}

if(more == 1){

# Decision - will p2 try to accept the increased wager?

if(p2\_method %in% c("auto", "non-card-counter")){

p2\_chance\_enh <- case\_when(

p2\_method == "auto" ~ p2\_chance,

p2\_method == "non-card-counter" ~ sum(full\_pack$id < p2$id) / 51

)

p2\_expected <- p2\_chance\_enh \* (wagered + extra \* 2) - (1 - p2\_chance\_enh) \* (wagered + extra \* 2)

accept = p2\_expected > (-wagered)

} else {

if(p2\_method == "ask"){

pm\_cat(paste0("You drew the ", p2$label, ". The other player raises the wager by ", extra, ". Do you accept? (Y/n)\n"))

input <- ""

while(!tolower(input) %in% c("y", "n")){

input <- readLines(n = 1)

}

accept = (tolower(input) == "y")

} else {

# random acceptance:

accept = as.logical(runif(1, 0 , 1) > 0.5)

}

}

if(accept & p2\_counters >= extra){

# the bet is on:

p1\_counters <- p1\_counters - extra

p2\_counters <- p2\_counters - extra

wagered <- wagered + extra \* 2

pm\_cat("p2 accepts the added wager.\n")

} else {

# or else, p2 gives up:

p1\_counters <- p1\_counters + wagered

wagered <- 0

if(verbose){

pm\_cat("p2 did not accept the added wager.\n")

}

}

}

# Who won?

if(p1$id > p2$id){

p1\_counters <- p1\_counters + wagered

if(verbose){

pm\_cat(paste0(p1$label, " beats ", p2$label, "; p1 wins.\n"))

}

} else {

p2\_counters <- p2\_counters + wagered

if(verbose){

pm\_cat(paste0(p1$label, " loses to ", p2$label, "; p2 wins.\n"))

}

}

known\_pack <- tmp2$remaining\_pack

if(nrow(known\_pack) < 2){

pm\_cat("Getting a fresh pack of cards.\n")

known\_pack <- pack

}

return(list(

p1\_counters = p1\_counters,

p2\_counters = p2\_counters,

known\_pack = known\_pack))

}

Having defined the functionality of a single hand, I then wrote a function for the main cycle of a game of user-specified rounds. This is basically a wrapper around pm\_hand, making it user-friendly for a player with sufficient messages to the console.

#-----------------------Human versus computer---------------------

#' Play a human v computer game of Persian Monarchs

#'

#' @param human number of chips human starts with

#' @param computer number of chips computer starts with

#' @param number of rounds to play (both players get a turn in a round)

persian\_monarchs\_hvc <- function(human = 100, computer = 100, rounds = 10){

start\_pack <- data\_frame(

id = 1:52,

suit = rep(c("clubs", "diamonds", "hearts", "spades"), each = 13),

value = rep(c(2:10, "Jack", "Queen", "King", "Ace"), 4)

) %>%

mutate(label = paste(value, "of", suit))

known\_pack <- start\_pack

for(i in 1:rounds){

# Computer deals:

if(i != 1) {

cat("Press enter for next hand.\n")

readLines(n = 1)

}

message("Computer's deal.")

cp <- pm\_hand(human,

computer,

known\_pack,

p1\_method = "ask",

verbose = TRUE,

p1\_lab = "Human",

p2\_lab = "Computer")

human <- cp$p1\_counters

computer <- cp$p2\_counters

known\_pack <- cp$known\_pack

cat(paste0("You have ", human, " counters.\n"))

cat(paste0("There are ", nrow(known\_pack), " cards left in the pack.\n"))

# Human deals:

cat("Press enter for next hand.\n")

readLines(n = 1)

message("Your deal.")

cp <- pm\_hand(computer,

human,

known\_pack,

p2\_method = "ask",

verbose = TRUE,

p1\_lab = "Computer",

p2\_lab = "Human")

human <- cp$p2\_counters

computer <- cp$p1\_counters

known\_pack <- cp$known\_pack

cat(paste0("You have ", human, " counters.\n"))

cat(paste0("There are ", nrow(known\_pack), " cards left in the pack.\n"))

}

}

# Run an interactive game against the computer for three rounds

persian\_monarchs\_hvc(rounds = 3)

This lets us play the game interactively in the R console, with a typical three round game shown in the screenshot below:



As you can see, at the end of the three rounds I’d lost a total of one counter to the computer, playing a more or less ad hoc but sensible strategy of increasing the wager by substantial amounts if I draw a spade, less so for hearts, and minimal or nothing for the minor suits (and comparable strategy for acceptance).

**Simulating card game results**

It was fun to write a computer game in R, but this particular one isn’t really fascinating enough to play thousands of rounds to explore optimal strategy. To do that we want a function that lets a computer play against itself, so I adapt the persian\_monarchs\_hvc function to persian\_monarchs\_cvc (computer versus computer) as below. We can specify one of the three automated strategies for each player.

#-------------------------computer versus computer-------------------

#' Play a computer v computer game of Persian Monarchs

#'

#' @param c1 number of chips Computer One starts with

#' @param c2 number of chips Computer Two starts with

#' @param c1\_method one of "auto" (best choice given expected value) and "random" (coin flip) for

#' deciding whether Computer One offers or accepts an increased wager

#' @param c2\_method one of "auto" (best choice given expected value) and "random" (coin flip) for

#' deciding whether Computer Two offers or accepts an increased wager

#' @param number of rounds to play (both players get a turn as dealer in a round, so a round is two hands)

#' @param verbose whether or not to print results to the screen

persian\_monarchs\_cvc <- function(c1 = 100, c2 = 100,

c1\_method = c("auto", "random", "non-card-counter"),

c2\_method = c("auto", "random", "non-card-counter"),

rounds = 10, verbose = TRUE){

c1\_method <- match.arg(c1\_method)

c2\_method <- match.arg(c2\_method)

start\_pack <- data\_frame(

id = 1:52,

suit = rep(c("clubs", "diamonds", "hearts", "spades"), each = 13),

value = rep(c(2:10, "Jack", "Queen", "King", "Ace"), 4)

) %>%

mutate(label = paste(value, "of", suit))

known\_pack <- start\_pack

for(i in 1:rounds){

# c2 deals:

if(verbose){message("Computer Two's deal.")}

cp <- pm\_hand(c1,

c2,

known\_pack,

p1\_method = c1\_method,

p2\_method = c2\_method,

verbose = verbose,

p1\_lab = "Computer One",

p2\_lab = "Computer Two")

c1 <- cp$p1\_counters

c2 <- cp$p2\_counters

known\_pack <- cp$known\_pack

if(verbose){

cat(paste0("Computer One has ", c1, " counters.\n"))

cat(paste0("There are ", nrow(known\_pack), " cards left in the pack.\n"))

}

# c1 deals:

if(verbose){

message("Computer One's deal.")

}

cp <- pm\_hand(c2,

c1,

known\_pack,

p1\_method = c2\_method,

p2\_method = c1\_method,

verbose = verbose,

p1\_lab = "Computer Two",

p2\_lab = "Computer One")

c1 <- cp$p2\_counters

c2 <- cp$p1\_counters

known\_pack <- cp$known\_pack

if(verbose){

cat(paste0("Computer One has ", c1, " counters.\n"))

cat(paste0("There are ", nrow(known\_pack), " cards left in the pack.\n"))

}

}

return(data\_frame(c1 = c1, c2 = c2))

}

# examples:

persian\_monarchs\_cvc(rounds = 3, verbose = TRUE)

persian\_monarchs\_cvc(rounds = 26, c1\_method = "auto", c2\_method = "auto", verbose = FALSE)

persian\_monarchs\_cvc(rounds = 26, c1\_method = "auto", c2\_method = "random", verbose = FALSE)

Now that we have this function, it’s a simple matter of playing many thousands of games between computer opponents with differing strategies. This sort of thing usually needs parallel processing (ie running simulations on as many processors as are available at once, rather than just one at a time) to happen in a reasonable period of time. These days, R has plenty of packages to enable this, even on Windows.

The code below does the simulations (and modelling and presentation of results) of what I’ve called the “optimal” strategy versus an identical strategy, a “no card counting” alternative, and making wager decisions at random.

#--------------------multiple simulations using parallel processing--------------------

# cluster with seven processors:

cl <- makeCluster(7)

# load the functionality we need onto the cluster:

clusterEvalQ(cl, {library(tidyverse)})

clusterExport(cl, c("persian\_monarchs\_cvc", "full\_pack", "pack", "pm\_hand", "pm\_draw"))

# number of simulations to do in each run. Takes about 90 seconds per run to do 10,000.

n <- 10000

# Run 1 - with both players using the optimal strategy

results\_auto <- parLapply(cl, 1:n, function(x){

persian\_monarchs\_cvc(rounds = 26, c1\_method = "auto", c2\_method = "auto", verbose = FALSE)

})

results\_auto\_df <- do.call("rbind", results\_auto) %>%

mutate(c2\_method = "Best strategy")

# Run 2 - computer two plays ok but without counting cards left in the pack

results\_ncc <- parLapply(cl, 1:n, function(x){

persian\_monarchs\_cvc(rounds = 26, c1\_method = "auto", c2\_method = "non-card-counter", verbose = FALSE)

})

results\_ncc\_df <- do.call("rbind", results\_ncc) %>%

mutate(c2\_method = "OK strategy but no card counting")

# Run 3 - computer two makes and accepts bets at random

results\_random <- parLapply(cl, 1:n, function(x){

persian\_monarchs\_cvc(rounds = 26, c1\_method = "auto", c2\_method = "random", verbose = FALSE)

})

results\_random\_df <- do.call("rbind", results\_random) %>%

mutate(c2\_method = "Random")

stopCluster(cl)

results\_combined <- results\_auto\_df %>%

rbind(results\_random\_df) %>%

rbind(results\_ncc\_df)

p <- results\_combined %>%

mutate(c2\_method = fct\_reorder(c2\_method, c2)) %>%

ggplot(aes(x = c2, fill = c2\_method, colour = c2\_method)) +

geom\_density(alpha = 0.5) +

#facet\_wrap(~c2\_method) +

labs(fill = "Wagering decisions:",

x = "Counters left at end of game",

colour = "Wagering decisions:",

y = paste("Density of results from", format(n, big.mark = ","), "simulations")) +

ggtitle("Results of two automated wagering strategies for Persian Monarchs",

"Counters left (having started with 100) after 26 rounds (two full packs) against a\nsound strategy")

print(p)

mod <- lm(c2 ~ c2\_method, data = results\_combined)

confint(mod)

**Reflection**

Astute readers will have noticed that the strategy I’ve sometimes called “optimal” is anything but. In fact, I’ve assumed away the chance of learning anything from the opponent’s own behaviour, whether a priori (for instance the reasonable assumption that they are more likely to accept or offer increased wagers with better cards) or after observing them for many rounds. So if we were seeking to create a Persian Monarchs master computer player, the algorithms above would be a sort of starting base case, to which we would need to add the ability to learn all sorts of tactics and strategies such as reading the opponent’s tendencies, swapping strategies oneself to confuse the opponent, bluffs and counter bluffs.

But that’s enough for now. In 2019 I’ll be extending this idea to some reflections on variants of Snakes and Ladders, and then to a classic 1970s text-based computer game that’s the granddaddy of *Civilization* and its turn-based strategic world-building ilk.