MAS6024 Assignment

210115996

Part 1

First, create a letter board matrix named lgrid. And then define two vector to store the green square's position and move direction for white square. The function called move_square takes the current square position as an argument and returns the square that the token moves to following the rules of the game. The function called is_edge_square is used to judge the current square whether it is on the edge of the board. The function called is_green_square is used to know the current square's color.

```
lgrid <- matrix(NA, nrow = 8, ncol = 8)
lgrid[1,] <- c("r", "l", "q", "s", "t", "z", "c", "a")
lgrid[2,] <- c("i", "v", "d", "z", "h", "l", "t", "p")
lgrid[3,] <- c("u", "r", "o", "y", "w", "c", "a", "c")
lgrid[4,] <- c("x", "r", "f", "n", "d", "p", "g", "v")
lgrid[5,] <- c("h", "j", "f", "f", "k", "h", "g", "m")
lgrid[6,] <- c("k", "y", "e", "x", "x", "g", "k", "i")
lgrid[7,] <- c("l", "q", "e", "q", "f", "u", "e", "b")
lgrid[8,] <- c("l", "s", "d", "h", "i", "k", "y", "n")

#create the vector of green square's position
green_position <- c(14, 23, 42, 51)

#create the move direction when choose un-edged white square
white_direction <- c(-1, 1, -7, 7, -8, 8, -9, 9)

is_edge_square <- function(current) {
  is_edge <- (current <= 8 || current >= 57 || current %% 8 == 0 || current %% 8 == 1)
  return(is_edge)
}

is_green_square <- function(current) {
  is_green <- current %in% green_position
  return(is_green)
}
move_square <- function(current) {
  next_square = 0
  if (is_edge_square(current)) {
    next_square <- sample(1:64, size = 1)
  } else {
    next_square <- current + sample(white_direction, size = 1)
  }
  return(next_square)
}
```

Part 2

a) My rules for deciding whether to add the letter to the player's collection if the token lands on a white square: 1.When the length of collection is smaller than 3, add the next letter directly. 2.When the length of collection equals to 3 and the next letter equals to one or more letters in the collection, add it. 3.When the length of collection equals to 4, estimate the string formed by letters in the collection. Depending on the format of the string, call the function named is_palindrome to choose different rule about forming the palindrome to decide whether to add the letter to the collection.

b) According my rules, when the length of the collection reach to 4, the collection will contain at least two same letters. And the final letter added to the collection will form a second pair of the same letters with another letter(unless there are already two pair of same letters in the collection before the final letter added). Perhaps we can estimate the probabilities of forming the palindrome with different prefix string, and remove some low-probability prefix cases. Which means we do not add the letter to the collection when the next letter will form the low-probability prefix string with current letters in the collection even it also can form palindrome finally.I think it can improve the strategy at the cost of increasing complexity.

Part 3

```r
## if current is a white square,
## choose whether to add the letter to the collection
choose_whether_add <-function(current, collection) {
  collect_length <- length(collection)
  letter <- lgrid[current]
  if (collect_length < 3) {
    return(TRUE)
  } else if (collect_length == 3){
    return(letter %in% collection)
  } else if (collect_length == 4){
    return(is_palindrome(letter, collection))
  }
}

is_palindrome <- function(letter, collection) {
  collection <- sort(collection)
  ## like "abcc"
  if (collection[1] != collection[2] && collection[2] != collection[3]) {
    return((letter == collection[1] || letter == collection[2]))
  }

  ## like "aabc"
  if (collection[1] == collection[2] && collection[2] != collection[3]
      && collection[3] != collection[4]) {
    return((letter == collection[3] || letter == collection[4]))
  }

  ## like aabb aaaa
  if (collection[1] == collection[2] && collection[3] == collection[4]) {
    return(TRUE)
  }

  ## like abbc abbb aaab
  return(letter == collection[1] || letter == collection[4])
}
```

```r
## if current is a green square
## the random events happen
green_events <- function(current, green_prob, collection) {
  ## random event happen according to the probability
  mode <- sample(0:1, size = 1, prob=c(green_prob, 1-green_prob))
  if(mode == 0) {
    collection <- c("f", "f", "h", "k")
  } else {
    letter <- lgrid[current]
    remove_index <- which(collection == letter)
    if(length(remove_index) != 0) {
      collection <- collection[-remove_index]
    }
  }
  return(collection)
}
## The code for the game
## start_position is a vector like(4, 4) means square D4
count_num_turns <- function(lgrid_matrix,start_position,green_prob, to_print) {
  num_turns <- 0
  current <- start_position[1] + (start_position[2] - 1) * 8
  collection <- vector()
  while(length(collection) < 5) {
    ## for green square
    if(is_green_square(current)) {
      if(to_print == T) cat ("current square = ", current, "it's a green square \n")
      collection <- green_events(current, green_prob, collection)
      current <- move_square(current)
    } else {
      ## for white square
      ## choose whether to add current square letter to the collection
      if(to_print == T) cat ("current square = ", current, "it's an white square \n")
      if(choose_whether_add(current, collection)) {
        collection <- append(collection, lgrid[current])
      }

      ## Is current square an edge square?
      if(is_edge_square(current)) {
        if(to_print == T) cat ("it's an edge square!\n")
        current <- move_square(current)
      } else {
        current <- move_square(current)
      }
    }
    num_turns <- num_turns + 1
  }
  if(to_print == T) {
    cat ("the collection result:", collection, "\n")
    cat ("the number of turns:", num_turns, "\n")
  }
  return(num_turns)
}
```

```r
## Test the game with start point (4,4) and p = 0.95
count_num_turns(lgrid, c(4, 4), 0.95, to_print = T)
```

```
## current square =  28 it's an white square
## current square =  27 it's an white square
## current square =  36 it's an white square
## current square =  45 it's an white square
## current square =  38 it's an white square
## current square =  29 it's an white square
## current square =  36 it's an white square
## current square =  37 it's an white square
## current square =  46 it's an white square
## current square =  37 it's an white square
## current square =  29 it's an white square
## current square =  36 it's an white square
## current square =  35 it's an white square
## current square =  34 it's an white square
## current square =  43 it's an white square
## current square =  50 it's an white square
## current square =  57 it's an white square
## it's an edge square!
## current square =  46 it's an white square
## current square =  54 it's an white square
## current square =  62 it's an white square
## it's an edge square!
## current square =  32 it's an white square
## it's an edge square!
## current square =  45 it's an white square
## current square =  38 it's an white square
## current square =  46 it's an white square
## current square =  55 it's an white square
## current square =  63 it's an white square
## it's an edge square!
## current square =  62 it's an white square
## it's an edge square!
## current square =  4 it's an white square
## it's an edge square!
## current square =  55 it's an white square
## current square =  47 it's an white square
## current square =  46 it's an white square
## current square =  37 it's an white square
## current square =  29 it's an white square
## current square =  22 it's an white square
## current square =  15 it's an white square
## current square =  22 it's an white square
## current square =  21 it's an white square
## current square =  28 it's an white square
## the collection result: n y d d n
## the number of turns: 38
```

```
## [1] 38
```

Part 4

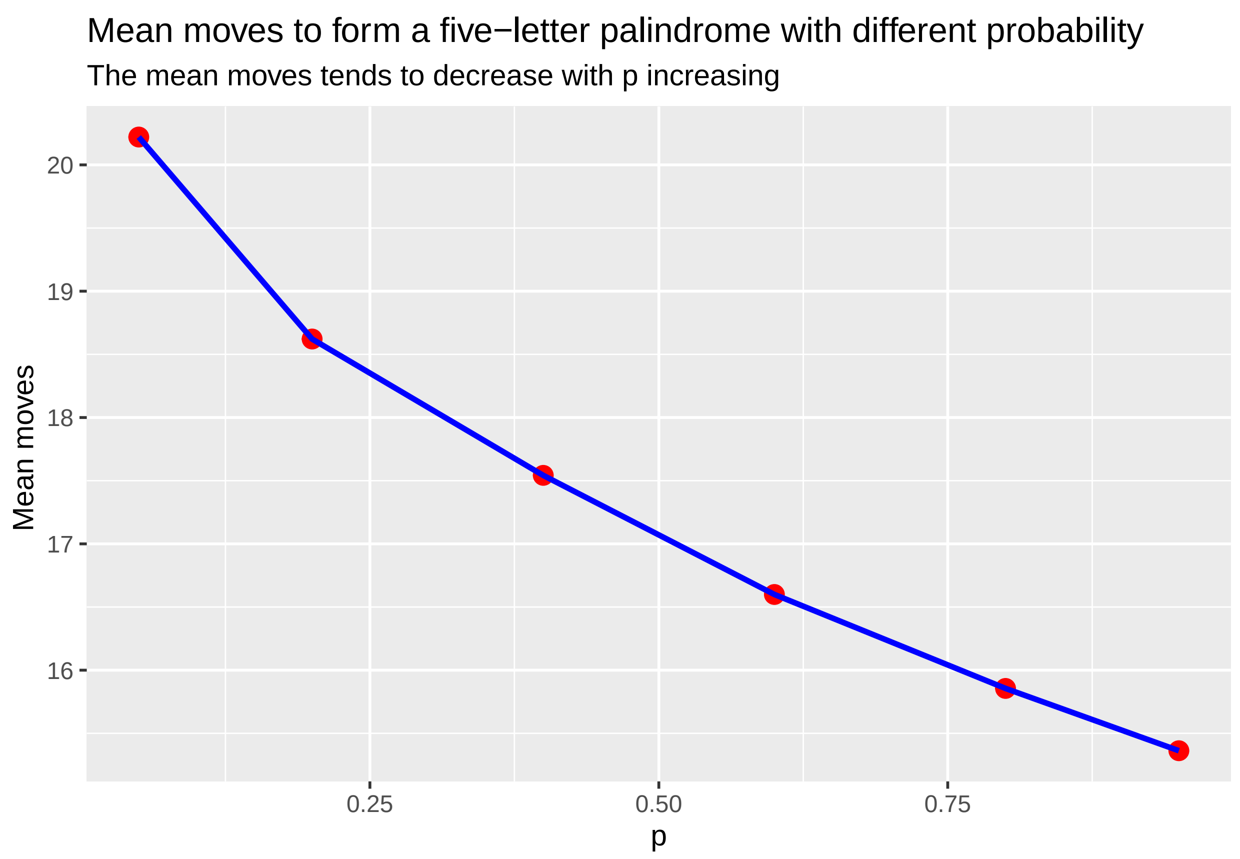
The position of the square D4 in the matrix is (4, 4), we change the parameter p from 0.05 to 0.95, and replicate the game function 10000 times with each probability. Then we calculate the mean of moves with different p values and compare the results by a line plot.

According to the plot, we can easily find that the number of moves required to finish the game decreases as the value of p increases.

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
mean_dist_of_num_turns <- function(replicate_time = 10000, sp, green_prob) {
  dist_of_num_turns <-
    replicate(replicate_time, count_num_turns(lgrid, sp, green_prob, to_print = F))
  return(mean(dist_of_num_turns))
}

mean_one <- mean_dist_of_num_turns(sp = c(4,4), green_prob = 0.05)
mean_two <- mean_dist_of_num_turns(sp = c(4,4), green_prob = 0.20)
mean_three<- mean_dist_of_num_turns(sp = c(4,4), green_prob = 0.40)
mean_four <- mean_dist_of_num_turns(sp = c(4,4), green_prob = 0.60)
mean_five <- mean_dist_of_num_turns(sp = c(4,4), green_prob = 0.80)
mean_six <- mean_dist_of_num_turns(sp = c(4,4), green_prob = 0.95)

probs <- c(0.05, 0.20, 0.40, 0.60, 0.80, 0.95)
mean_values <- c(mean_one, mean_two, mean_three, mean_four, mean_five, mean_six)
df_part4 <- data.frame(prob = probs, mean_value = mean_values)

ggplot(data = df_part4, aes(x = prob, y = mean_value)) +
  geom_point(size = 3, colour = "red") +
  geom_line(size = 1, colour = "blue") +
  labs(x = "p", y = "Mean moves ",
       title = "Mean moves to form a five-letter palindrome with different probability",
       subtitle = "The mean moves tends to decrease with p increasing")
```



Part 5

First, replicate the game 10000 times for starting points D4 with p = 0.95 and F6 with p = 0.05 and calculate the mean number of each moves distribution. Then we call quantile function to estimate the probability distribution of the number of moves needed to complete the game. And then we create boxplot for the results to compare the distributions of numbers of moves required to complete the game for two starting points with different p values.

```
dist_of_num_turns_d4 <- replicate(10000, count_num_turns(lgrid, c(4,4), green_prob=0.95, to_print = F))
dist_of_num_turns_f6 <- replicate(10000, count_num_turns(lgrid, c(6,6), green_prob=0.05, to_print = F))


mean_d4 <- mean(dist_of_num_turns_d4)
mean_f6 <- mean(dist_of_num_turns_f6)
cat("the mean moves of square D4 with p = 0.95:", mean_d4, "\n")
```

```
## the mean moves of square D4 with p = 0.95: 15.5229
```

```
cat("the mean moves of square F6 with p = 0.05:", mean_f6, "\n")
```

```
## the mean moves of square F6 with p = 0.05: 19.2435
```

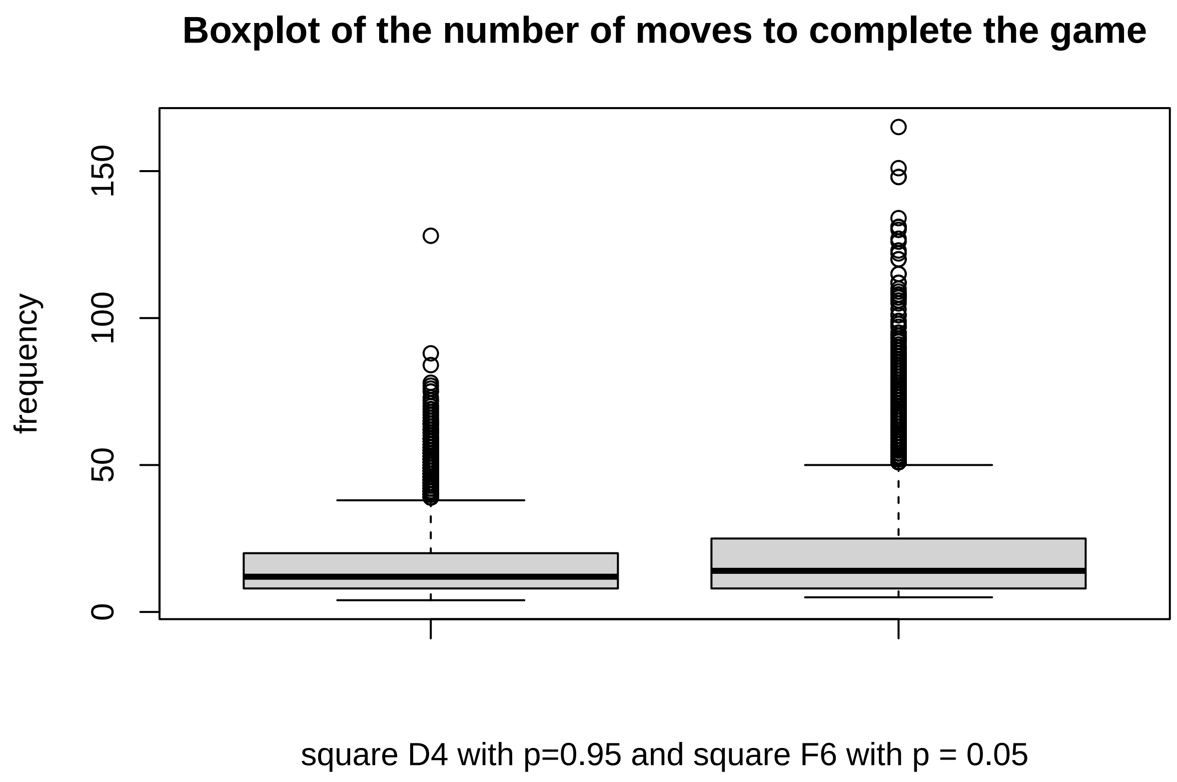
```
quantile(dist_of_num_turns_d4, seq(0.1, 1, 0.1))
```

```
##  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##    5    7    8   10   12   15   18   23   30  128
```

```
quantile(dist_of_num_turns_f6, seq(0.1, 1, 0.1))
```

```
##  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##    6    7    9   11   14   18   22   28   39  165
```

```
boxplot(dist_of_num_turns_d4, dist_of_num_turns_f6,
        xlab="square D4 with p=0.95 and square F6 with p = 0.05",
        ylab="frequency",
        main="Boxplot of the number of moves to complete the game")
```



According to the coding results above, we can find that the means of numbers of moves for two starting points are different. And in the range of 60 to 100 percent, it's clear that the mean number of moves of F6 point increases faster. In the box plot we can see this tendency too, the second box has a larger third quartile and larger maximum, and the second box contains more diffuse outliers. So we can conclude that the probability distributions of the number of moves required to complete the game are not identical for the starting points D4 with p = 0.95 and F6 with p = 0.05.

We can also create histograms and scatter plots to estimate the distributions of two starting points. I think they can enable me to answer the question.

Part 6

According to the question, we create to vectors to store the number of moves results first. Then we can use T-test to assess the evidence that $E(X_A)=E(X_B)$.

```
A <- c(25, 13, 16, 24, 11, 12, 24, 26, 15, 19, 34)
B <- c(35, 41, 23, 26, 18, 15, 33, 42, 18, 47, 21, 26)
```

```r
## Use Shapiro-Wilk Normality test to prove that A and B are normally distributed first
shapiro.test(A)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  A
## W = 0.9289, p-value = 0.3999
```

```r
shapiro.test(B)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  B
## W = 0.93139, p-value = 0.3951
```

```r
## Use homogeneity of variance test to prove that A and B are homogenous
var.test(A, B)
```

```
##
##  F test to compare two variances
##
## data:  A and B
## F = 0.46405, num df = 10, denom df = 11, p-value = 0.2371
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.1316216 1.7007204
## sample estimates:
## ratio of variances
##          0.4640547
```

```r
## T test
t.test(A, B, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  A and B
## t = -2.3075, df = 21, p-value = 0.0313
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -16.8086178  -0.8732004
## sample estimates:
## mean of x mean of y
##  19.90909  28.75000
```

According to the above t-test results we can find that the p-value of t-test is smaller than 0.05(p-value $<$ 0.05), we can assume that the expectations of A and B are different. In conclusion, $E(X_A) \neq E(X_B)$.