Stat301 Final Capstone

Dan Mariani

2024-12-16

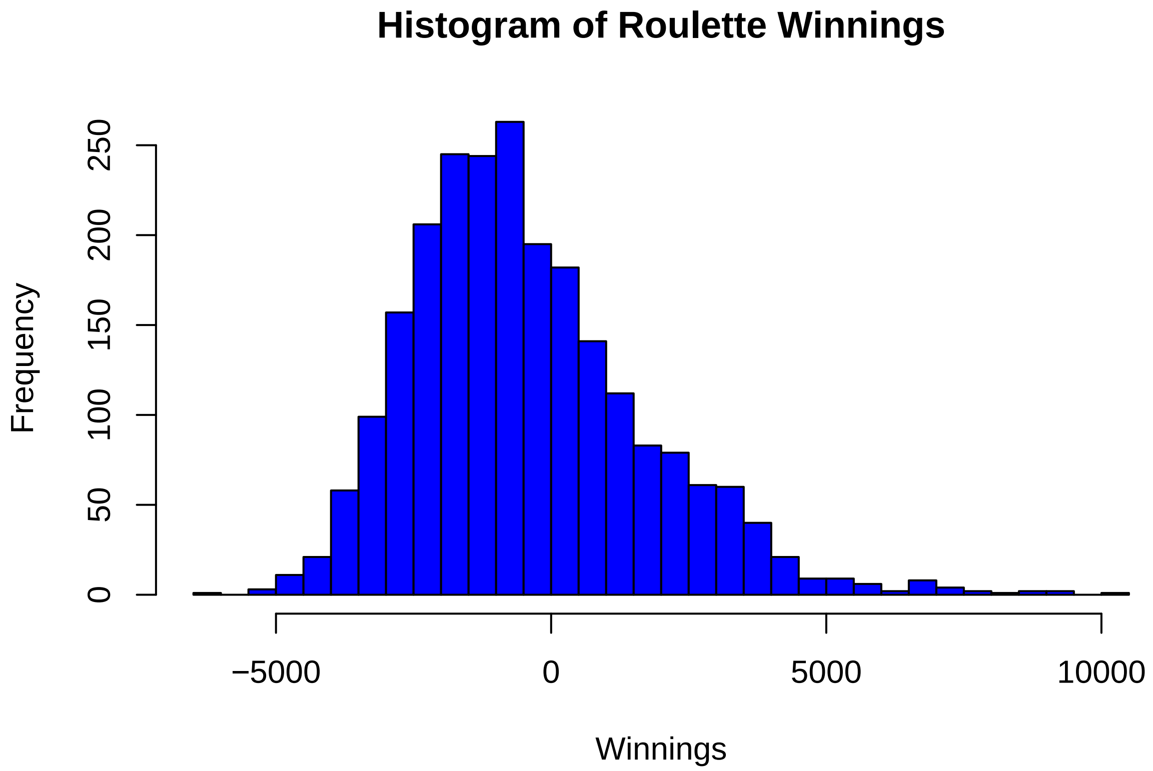
Part 1: Player Information

```r
set.seed(123)

# getting player information from provided distributions
number_players <- 8000
play_time <- rgamma(number_players, 40, 0.2)/60  #play time calculated in hours to multiply with rates
game_counts <- rmultinom(1, number_players, c(0.3, 0.05, 0.05, 0.1, 0.4, 0.1))  #calculates how mant pl
game_types <- rep(c("Roulette", "Baccarat", "Keno", "4-Reel", "5-Reel", "Craps"),
    game_counts)
games_per_hour <- c(Roulette = 50, Baccarat = 50, Keno = 10, `4-Reel` = 400, `5-Reel` = 400,
    Craps = 100)

# putting player info into list
players <- vector("list", number_players)
for (i in 1:number_players) {
    time_spent <- play_time[i]
    game_type <- game_types[i]
    games_played <- round(time_spent * games_per_hour[game_type])
    players[[i]] <- list(time_spent = time_spent, game_type = game_type, games_played = games_played,
        result = 0, bet_amount = 0)
}

# splitting players by the game they are playing
players_split <- split(players, sapply(players, function(x) x$game_type))
```

Roulette:

```r
#new function for roulette to create roulette table
#same logic as original roulette function with slightly different parameters

roulette <- function(bet_type, bet_size, casino_outcome) {
  #possible bet_types and outcomes

  payout_multipliers <- list(
    "single_number" = 35,
    "even_odd" = 1,
    "red_black" = 1,
    "low_high" = 1,
    "columns" = 2,
    "dozens" = 2,
    "line" = 5,
    "five_number" = 6,
    "corner" = 8,
    "street" = 11,
    "split" = 17
  )

  #changes payouts and functions based on bet_type
  calculate_winnings <- switch(bet_type,

    "single_number" = function() {
      player_pick <- sample(1:38, 1)
      if (player_pick == casino_outcome) payout_multipliers[[bet_type]] * bet_size else -bet_size
    },

    "even_odd" = function() {
      even_numbers <- seq(2, 36, 2)
      odd_numbers <- seq(1, 35, 2)
      player_pick <- sample(c("even", "odd"), 1)
      if ((player_pick == "even" & casino_outcome %in% even_numbers) |
          (player_pick == "odd" & casino_outcome %in% odd_numbers)) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "red_black" = function() {
      black_numbers <- c(1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34, 36)
      red_numbers <- c(2, 4, 6, 8, 10, 11, 13, 15, 17, 20, 22, 24, 26, 28, 29, 31, 33, 35)
      player_pick <- sample(c("red", "black"), 1)
      if ((player_pick == "black" & casino_outcome %in% black_numbers) |
          (player_pick == "red" & casino_outcome %in% red_numbers)) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "low_high" = function() {
      low_numbers <- 1:18
      high_numbers <- 19:36
      player_pick <- sample(c("low", "high"), 1)
      if ((player_pick == "low" & casino_outcome %in% low_numbers) |
          (player_pick == "high" & casino_outcome %in% high_numbers)) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "columns" = function() {
      columns <- list(c1 = seq(1, 34, 3), c2 = seq(2, 35, 3), c3 = seq(3, 36, 3))
```

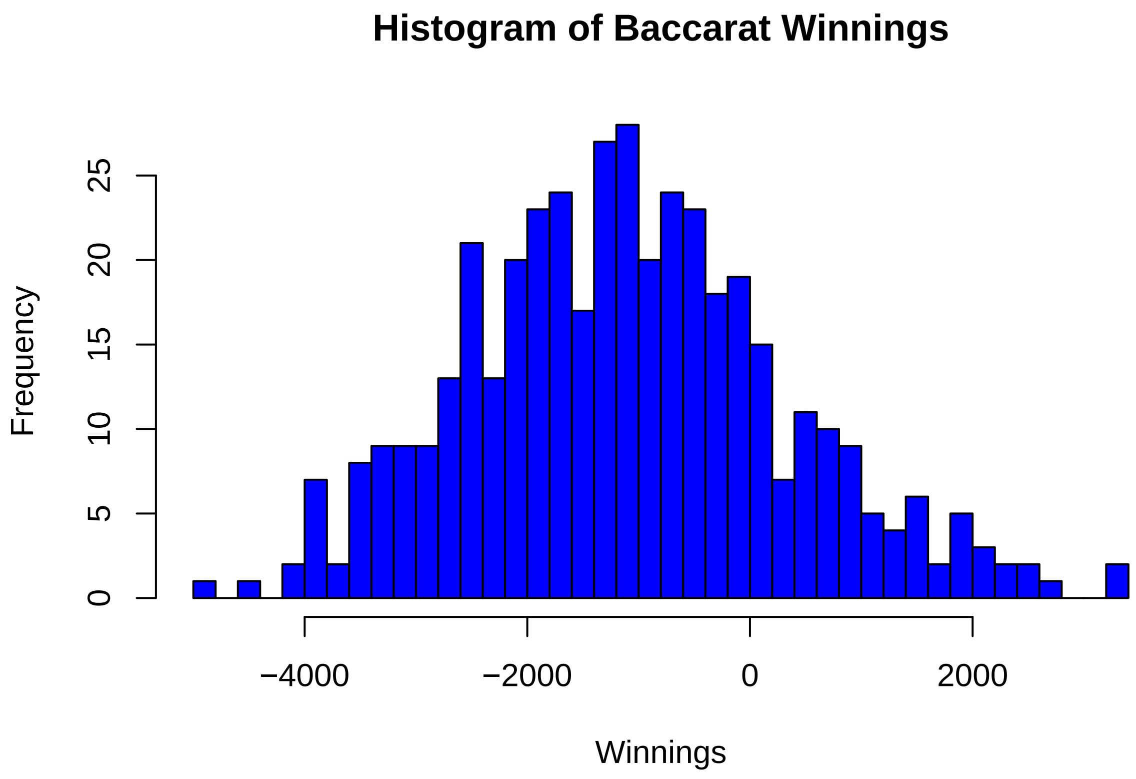
```
      player_pick <- sample(1:3, 1)
      if(casino_outcome %in% columns[[player_pick]]) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "dozens" = function() {
      dozens <- list(d1 = 1:12, d2 = 13:24, d3 = 25:36)
      player_pick <- sample(1:3, 1)
      if (casino_outcome %in% dozens[[player_pick]]) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "line" = function() {
      line_start <- sample(seq(1, 31, by = 3), 1)
      if (casino_outcome >= line_start & casino_outcome <= line_start + 5) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "five_number" = function() {
      five_number_bet <- c(37, 38, 1, 2, 3)
      if (casino_outcome %in% five_number_bet) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "corner" = function() {
      corner_start <- sample(c(1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31), 1)
      if (casino_outcome %in% c(corner_start, corner_start + 1, corner_start + 3, corner_start + 4)) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },

    "street" = function() {
      street_start <- sample(seq(1, 34, by = 3), 1)
      if (casino_outcome >= street_start & casino_outcome <= street_start + 2) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    },
    "split" = function() {
      split_bet <- matrix(c(1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31), ncol = 2)
      selected_split <- split_bet[sample(1:nrow(split_bet), 1), ]
      if (casino_outcome %in% selected_split) {
        payout_multipliers[[bet_type]] * bet_size
      } else {
        -bet_size
      }
    }
  )

  #calculates winnings for roulette
  total_winnings <- calculate_winnings()
  return(total_winnings)
}

#function to calculate results for roulette table (4 tables with 8 players each)
roulette_table <- function(roulette_players, number_tables = 4, max_players = 8) {
  #give each player id
  for (i in seq_along(roulette_players)) {
    roulette_players[[i]]$id <- i
  }

  #determine active players and players that will be cycled in
  active_players <- roulette_players[1:min(number_tables * max_players, length(roulette_players))]
  waiting_players <- roulette_players[-(1:length(active_players))]

  #split active players into the 4 tables
  table_indices <- rep(1:number_tables, length.out = length(active_players))
  tables <- split(active_players, table_indices)

  #define possible roulette outcomes
  roulette_numbers <- 1:38

  #loop to simulate through all roulette players until no players have any remaining games to play
  while (any(sapply(roulette_players, function(x) x$games_played > 0))) {
    for (table_index in seq_along(tables)) {
      table <- tables[[table_index]]

      #generate casiono roulette spin for each table
      casino_outcome <- sample(roulette_numbers, 1)

      for (i in seq_along(table)) {
        player <- table[[i]]
        if (player$games_played > 0) {
          #randomly sample bet type and size for each player in each table

          bet_type <- sample(c("single_number", "red_black", "even_odd", "low_high",
                               "columns", "dozens", "corner", "street"), 1)
          bet_size <- sample(10:100, 1) #random bet size between 10 and 100

          #simulate the roulette game for each player based on above roulette function
          player_result <- roulette(bet_type, bet_size, casino_outcome)
          #update each players outcome
          player$result <- player$result + player_result
          player$bet_amount <- player$bet_amount + bet_size
          player$games_played <- player$games_played - 1

          #replace players that have played all their games with new players
          if (player$games_played <= 0 & length(waiting_players) > 0) {
            table[[i]] <- waiting_players[[1]]
            waiting_players <- waiting_players[-1]
          } else {
            table[[i]] <- player
          }

          #update players and tables
          player_id <- player$id
          roulette_players[[player_id]] <- player
        }
      }
      tables[[table_index]] <- table
    }
  }

  #histogram of winnings
    winnings <- sapply(roulette_players, function(x) x$result)
    hist(winnings,
      breaks = 50,
      col = "blue",
      border = "black",
      main = "Histogram of Roulette Winnings",
      xlab = "Winnings",
      ylab = "Frequency")

  #calculate total results and money spent for all players and find house advantage
  total_result <- sum(sapply(roulette_players, function(x) x$result))
  total_bet_amount <- sum(sapply(roulette_players, function(x) x$bet_amount))
  house_advantage <- -(total_result / total_bet_amount)
  return(list(total_result = total_result,
              total_bet_amount = total_bet_amount,
              house_advantage = house_advantage))
}

#assign roulette players from list
roulette_players <- players_split[["Roulette"]]

#get results
roulette_result <- roulette_table(roulette_players)
```



Baccarat:

```r
# new function for baccarat to create baccarat table same logic as original
# baccarat function with slightly different parameters

baccarat <- function(bet_type, bet_size, casino_outcome) {

    # function to calculate hand values
    handval <- function(cards) {
        sum(cards)%%10
    }

    # initialize and shuffle deck
    deck <- rep(c(rep(1:9, 4), rep(0, 16)), 8)  #8 decks used
    shuffle <- sample(deck, length(deck), replace = FALSE)

    # deal cards to player and banker
    playerhand <- shuffle[1:2]
    bankerhand <- shuffle[3:4]
    player3 <- banker3 <- NULL

    # calculate hand values
    playerval <- handval(playerhand)
    bankerval <- handval(bankerhand)

    # check for winner
    if (!(playerval >= 8 | bankerval >= 8)) {

        # player 3rd card
        if (playerval <= 5) {
            player3 <- shuffle[5]
            playerval <- handval(c(playerhand, player3))  #recalculate hand
        }
        # banker 3rd card
        if (playerval %in% c(6, 7)) {
            if (bankerval <= 5)
                banker3 <- shuffle[6]
        } else if (player3 != 8) {
            if (bankerval <= 2)
                banker3 <- shuffle[6] else if (bankerval == 3)
                banker3 <- shuffle[6] else if (bankerval == 4 & player3 %in% c(2, 3, 4, 5, 6, 7))
                banker3 <- shuffle[6] else if (bankerval == 5 & player3 %in% c(4, 5, 6, 7))
                banker3 <- shuffle[6] else if (bankerval == 6 & player3 %in% c(6, 7))
                banker3 <- shuffle[6]
        }
        bankerval <- handval(c(bankerhand, banker3))  #recalculate hand
    }

    # determine winner
    outcome <- if (playerval > bankerval)
        "player" else if (bankerval > playerval)
        "banker" else "tie"

    # define payouts for each bet type
    payouts <- list(player = ifelse(outcome == "player", bet_size, -bet_size), banker = ifelse(outcome =
        "banker", bet_size * 0.95, -bet_size), tie = ifelse(outcome == "tie", bet_size *
        8, -bet_size))

    # calculate winnings based on bet type
    winnings <- payouts[[bet_type]]
    return(winnings)
}

# function to calculate results for baccarat table (2 tables with 6 players
# each)
baccarat_table <- function(baccarat_players, number_tables = 2, max_players = 6) {
    # assign player id
    for (i in seq_along(baccarat_players)) {
        baccarat_players[[i]]$id <- i
    }

    # determine active and next players to cycle through
    active_players <- baccarat_players[1:min(number_tables * max_players, length(baccarat_players))]
    waiting_players <- baccarat_players[-(1:length(active_players))]

    # split players into 2 tables
    table_indices <- rep(1:number_tables, length.out = length(active_players))
    tables <- split(active_players, table_indices)
    # loop to simulate outcomes for every player for every round until no
    # players have anymore rounds to play
    while (any(sapply(baccarat_players, function(x) x$games_played > 0))) {
        for (table_index in seq_along(tables)) {
            table <- tables[[table_index]]

            for (i in seq_along(table)) {
                player <- table[[i]]
                if (player$games_played > 0) {
                  # selects random bet type and bet size between 10 and 100
                  bet_type <- sample(c("player", "banker", "tie"), 1)
                  bet_size <- sample(10:100, 1)

                  # simulates baccarat game based on previous function
                  player_result <- baccarat(bet_type, bet_size, NULL)

                  # update results for each player
                  player$result <- player$result + player_result
                  player$bet_amount <- player$bet_amount + bet_size
                  player$games_played <- player$games_played - 1

                  # cycle through players
                  if (player$games_played <= 0 & length(waiting_players) > 0) {
                    table[[i]] <- waiting_players[[1]]
                    waiting_players <- waiting_players[-1]
                  } else {
                    table[[i]] <- player
                  }

                  # update lists
                  player_id <- player$id
                  baccarat_players[[player_id]] <- player
                }
            }
            tables[[table_index]] <- table
        }
    }

    # histogram of winnings
    winnings <- sapply(baccarat_players, function(x) x$result)
    hist(winnings, breaks = 50, col = "blue", border = "black", main = "Histogram of Baccarat Winnings"
        xlab = "Winnings", ylab = "Frequency")

    # calculate total results and house advantage
    total_result <- sum(sapply(baccarat_players, function(x) x$result))
    total_bet_amount <- sum(sapply(baccarat_players, function(x) x$bet_amount))
    house_advantage <- -(total_result/total_bet_amount)

    return(list(total_result = total_result, total_bet_amount = total_bet_amount,
        house_advantage = house_advantage))
}

# split players playing baccarat
```

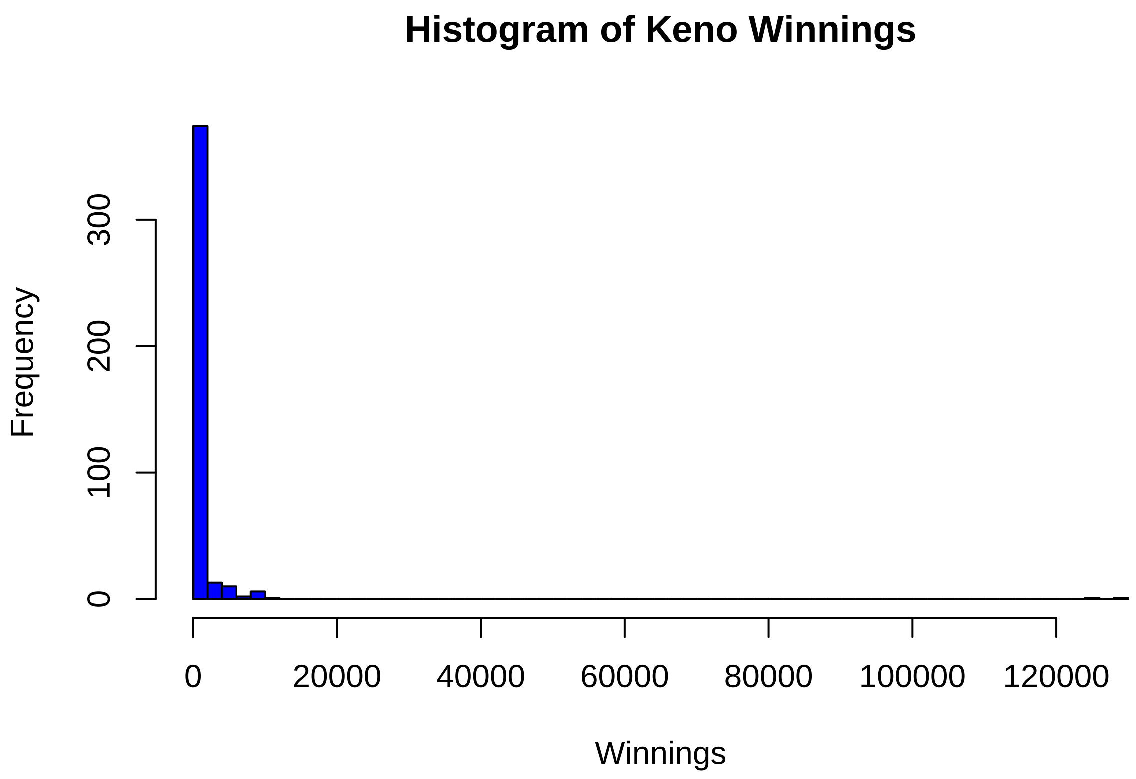
```
baccarat_players <- players_split[["Baccarat"]]

# simulate results
baccarat_result <- baccarat_table(baccarat_players)
```



Keno:

```
# new function for keno to create keno room

# selects player numbers and sees how many match with casino numbers
keno <- function(bet_size, casino_numbers) {
    player_numbers <- sample(1:80, 8)
    catch <- sum(player_numbers %in% casino_numbers)
    payout_amounts <- c(`0` = 0, `1` = 0, `2` = 0, `3` = 0, `4` = 0, `5` = 9, `6` = 90,
        `7` = 1500, `8` = 20000)
    # returns correct payout based on matching numbers (catch)
    payout <- payout_amounts[as.character(catch)]
    total <- bet_size * payout
    return(total)
}

# function to calculate results for keno room (1 room with 100 players)
keno_room <- function(keno_players, number_rooms = 1, max_players = 100) {
    # assign player ids
    for (i in seq_along(keno_players)) {
        keno_players[[i]]$id <- i
```

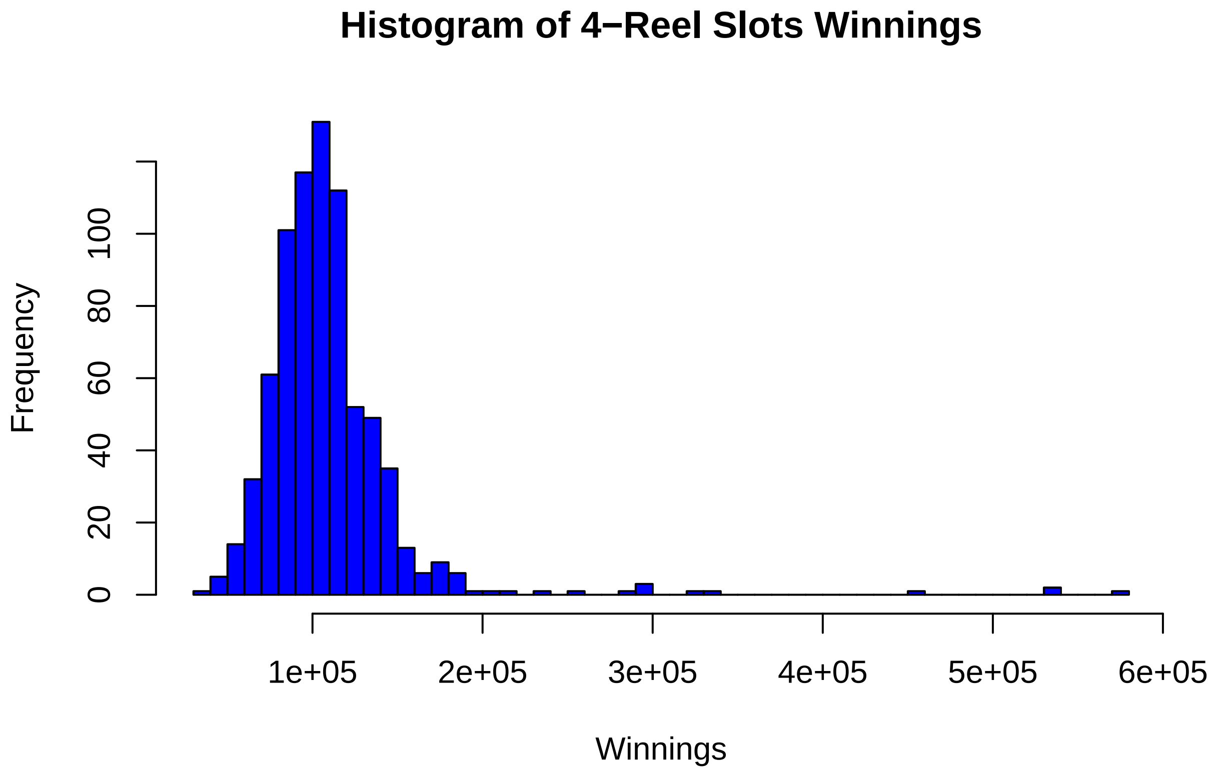
```r
}

active_players <- keno_players[1:min(number_rooms * max_players, length(keno_players))]
waiting_players <- keno_players[-(1:length(active_players))]

# split players into rooms
room_indices <- rep(1:number_rooms, length.out = length(active_players))
rooms <- split(active_players, room_indices)

keno_numbers <- 1:80  #indicates numbers you can draw in keno

# keeps looping until no players have any rounds left to play
while (any(sapply(keno_players, function(x) x$games_played > 0))) {
    for (room_index in seq_along(rooms)) {
        room <- rooms[[room_index]]

        # randomly generated casino numbers
        casino_numbers <- sample(keno_numbers, 20)

        for (i in seq_along(room)) {
            player <- room[[i]]
            if (player$games_played > 0) {
              bet_size <- sample(10:100, 1)  #chooses random bet size between 10 and 100

              # simulates the keno game based on previous function
              player_result <- keno(bet_size, casino_numbers)

              # calculates player results
              player$result <- player$result + player_result
              player$bet_amount <- player$bet_amount + bet_size
              player$games_played <- player$games_played - 1

              # replaces finished players with waiting players
              if (player$games_played <= 0 & length(waiting_players) > 0) {
                room[[i]] <- waiting_players[[1]]
                waiting_players <- waiting_players[-1]
              } else {
                room[[i]] <- player
              }

              # update lists
              player_id <- player$id
              keno_players[[player_id]] <- player
            }
        }
        rooms[[room_index]] <- room
    }
}

# histogram of winnings
winnings <- sapply(keno_players, function(x) x$result)
hist(winnings, breaks = 50, col = "blue", border = "black", main = "Histogram of Keno Winnings",
    xlab = "Winnings", ylab = "Frequency")
    # calculates total results and house advantage
    total_result <- sum(sapply(keno_players, function(x) x$result))
    total_bet_amount <- sum(sapply(keno_players, function(x) x$bet_amount))
    house_advantage <- (total_bet_amount - total_result)/total_bet_amount

    return(list(total_result = total_result, total_bet_amount = total_bet_amount,
        house_advantage))

}

# select keno players from list
keno_players <- players_split[["Keno"]]

# get results from function
keno_result <- keno_room(keno_players)
```



4-Reel Slots:

```
# new function for 4-rrel slots to create 4-reel slot machines same logic as
# original 4-reel slots function with slightly different parameters

four_reel_slots <- function(num_lines, bet_size_per_line, casino_outcome = NULL) {
    # define reels
    sample_space_1 <- list(6, 2, 1, 3, 5, 2, 3, 4, 5, 2, 4, 1, 5, 2, 3, 4, 5, 2,
        3, 2)
```

```r
    sample_space_2 <- list(6, 4, 1, 3, 4, 1, 4, 5, 1, 2, 3, 4, 1, 5, 4, 1, 2, 4,
        1, 4)
    sample_space_3 <- list(6, 2, 1, 3, 2, 3, 2, 3, 1, 3, 5, 2, 4, 3, 1, 3, 2, 3,
        2, 3)
    sample_space_4 <- list(6, 1, 3, 4, 3, 1, 3, 2, 3, 1, 2, 4, 3, 1, 3, 2, 3, 1,
        3, 4)

    # generate a casino outcome (numbers seen on reels)
    if (is.null(casino_outcome)) {
        slot_outcome <- function(space) {
            index <- sample(1:20, 1)
            left <- ifelse(index == 1, 20, index - 1)
            right <- ifelse(index == 20, 1, index + 1)
            list(space[[left]], space[[index]], space[[right]])
        }
        casino_outcome <- list(slot_outcome(sample_space_1), slot_outcome(sample_space_2),
            slot_outcome(sample_space_3), slot_outcome(sample_space_4))
    }

    # defines paylines
    check_payout <- function(pay_line) {
        sym1 <- pay_line[[1]]
        sym2 <- pay_line[[2]]
        sym3 <- pay_line[[3]]
        sym4 <- pay_line[[4]]
        # creates payout logic to check payouts
        if (sym1 == 6 & sym2 == 6 & sym3 == 6 & sym4 == 6) {
            return(5000)
        } else if ((sym1 == 6 & sym2 == 6 & sym3 == 6) | (sym1 == 5 & sym2 == 5 & sym3 ==
            5) | (sym1 == 4 & sym2 == 4 & sym3 == 4 & sym4 == 4)) {
            return(150)
        } else if ((sym1 == 3 & sym2 == 3 & sym3 == 3 & sym4 == 3) | (sym1 == 2 & sym2 ==
            2 & sym3 == 2 & sym4 == 2) | (sym1 == 1 & sym2 == 1 & sym3 == 1 & sym4 ==
            1)) {
            return(20)
        } else if (sym1 == 4 & sym2 == 4 & sym3 == 4) {
            return(18)
        } else if (sym1 == 3 & sym2 == 3 & sym3 == 3) {
            return(14)
        } else if ((sym1 == 2 & sym2 == 2 & sym3 == 2) | (sym1 == 1 & sym2 == 1 & sym3 ==
            1)) {
            return(10)
        } else if (sym1 == 1 & sym2 == 1) {
            return(5)
        } else if (sym1 == 1 & sym3 == 1 & sym2 != 1) {
            return(3)
        }
        return(0)
    }

    # defines paylines
    pay_line_1 <- list(casino_outcome[[1]][[1]], casino_outcome[[2]][[1]], casino_outcome[[3]][[1]],
        casino_outcome[[4]][[1]])
    pay_line_2 <- list(casino_outcome[[1]][[2]], casino_outcome[[2]][[2]], casino_outcome[[3]][[2]],
        casino_outcome[[4]][[2]])
    pay_line_3 <- list(casino_outcome[[1]][[3]], casino_outcome[[2]][[3]], casino_outcome[[3]][[3]],
        casino_outcome[[4]][[3]])

    # calculates payouts based on lines
    payout <- 0
    if (num_lines >= 1) {
        payout <- payout + check_payout(pay_line_1) * bet_size_per_line
    }
    if (num_lines >= 2) {
        payout <- payout + check_payout(pay_line_2) * bet_size_per_line
    }
    if (num_lines == 3) {
        payout <- payout + check_payout(pay_line_3) * bet_size_per_line
    }

    return(payout)
}

# function to calculate results for 4-reel slot machine (100 machines with 1
# player per machine)

four_reel_slot_machine <- function(slot_players, number_machines = 100) {
    # assign player id
    for (i in seq_along(slot_players)) {
        slot_players[[i]]$id <- i
    }

    # determine active and waiting players
    active_players <- slot_players[1:min(number_machines, length(slot_players))]
    waiting_players <- slot_players[-(1:length(active_players))]

    # split players across 100 machines
    machine_indices <- rep(1:number_machines, length.out = length(active_players))
    machines <- split(active_players, machine_indices)

    # loop through all players until no players have rounds left to play
    while (any(sapply(slot_players, function(x) x$games_played > 0))) {
        for (machine_index in seq_along(machines)) {
            machine <- machines[[machine_index]]

            for (i in seq_along(machine)) {
                player <- machine[[i]]

                if (player$games_played > 0) {
                  # randomly select number of lines and bet size
                  num_lines <- sample(1:3, 1)
                  bet_size_per_line <- sample(10:100, 1)

                  # use previous function to get game results
                  payout <- four_reel_slots(num_lines, bet_size_per_line)
                  # update each players winnings
                  player$result <- player$result + payout
                  player$bet_amount <- player$bet_amount + (num_lines * bet_size_per_line)
                  player$games_played <- player$games_played - 1

                  # cycle through waiting players
                  if (player$games_played <= 0 & length(waiting_players) > 0) {
                    machine[[i]] <- waiting_players[[1]]
                    waiting_players <- waiting_players[-1]
                  } else {
                    machine[[i]] <- player
                  }

                  # update lists
                  player_id <- player$id
                  slot_players[[player_id]] <- player
                }
            }
            machines[[machine_index]] <- machine
        }
    }

    # histogram of winnings
    winnings <- sapply(slot_players, function(x) x$result)
    hist(winnings, breaks = 50, col = "blue", border = "black", main = "Histogram of 4-Reel Slots Winni
        xlab = "Winnings", ylab = "Frequency")

    # calculate final results and house advantage
    total_result <- sum(sapply(slot_players, function(x) x$result))
    total_bet_amount <- sum(sapply(slot_players, function(x) x$bet_amount))
    house_advantage <- (total_bet_amount - total_result)/total_bet_amount

    return(list(total_result = total_result, total_bet_amount = total_bet_amount,
        house_advantage = house_advantage))
}



# splits players by game type
four_reel_players <- players_split[["4-Reel"]]

# runs function to get results
four_reel_result <- four_reel_slot_machine(four_reel_players)
```

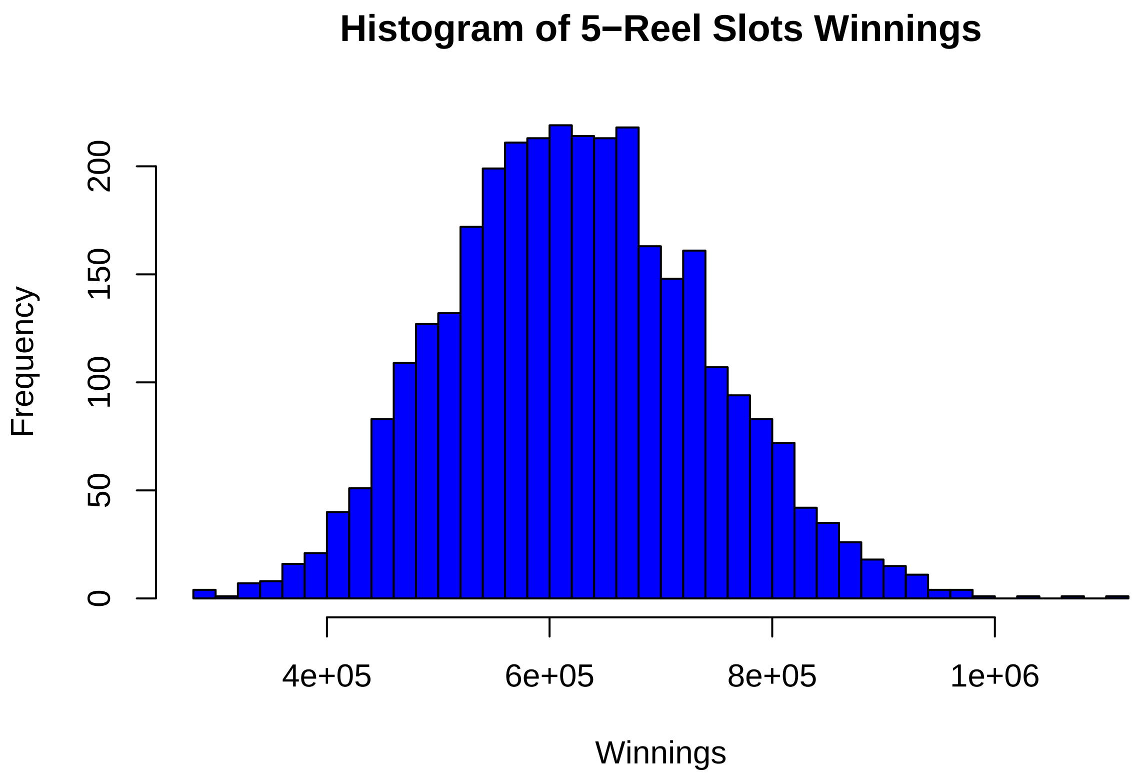


5-Reel Slots:

```
five_reel_slots <- function(num_lines, bet_size_per_line, casino_outcome = NULL) {

    # define the reels
    reel1 <- c(rep(1:12, 2), 13, 14, 15, rep(0, 2))
    reel2 <- reel1
    reel3 <- reel1
    reel4 <- reel1
    reel5 <- reel1

    if (!exists("progressive_jackpot_env", envir = .GlobalEnv)) {
        progressive_jackpot_env <- new.env()
        progressive_jackpot_env$progressive_jackpot <- 1000
        progressive_jackpot_env$target <- sample(1000:10000, 1)
        assign("progressive_jackpot_env", progressive_jackpot_env, envir = .GlobalEnv)
    }

    jackpot_env <- get("progressive_jackpot_env", envir = .GlobalEnv)

    # generate random casino outcome (players view of reels)
    if (is.null(casino_outcome)) {
        reels <- matrix(NA, nrow = 3, ncol = 5)
        reels[, 1] <- sample(reel1, 3, replace = FALSE)
        reels[, 2] <- sample(reel2, 3, replace = FALSE)
        reels[, 3] <- sample(reel3, 3, replace = FALSE)
```

```r
        reels[, 4] <- sample(reel4, 3, replace = FALSE)
        reels[, 5] <- sample(reel5, 3, replace = FALSE)
    } else {
        reels <- casino_outcome
    }

    total_wins <- 0
    payline_wins <- 0

    # define paylines to check payouts
    payline_patterns <- list(c(1, 1, 1, 1, 1), c(2, 2, 2, 2, 2), c(3, 3, 3, 3, 3),
        c(1, 2, 3, 2, 1), c(3, 2, 1, 2, 3), c(1, 1, 2, 1, 1), c(3, 3, 2, 3, 3), c(2,
            1, 1, 1, 2), c(2, 3, 3, 3, 2), c(1, 3, 3, 3, 1), c(3, 1, 1, 1, 3), c(1,
            1, 2, 3, 3), c(3, 3, 2, 1, 1), c(2, 3, 2, 1, 2), c(2, 1, 2, 3, 2))

    # add extra paylines
    if (num_lines == 30) {
        payline_patterns <- append(payline_patterns, list(c(1, 2, 1, 2, 1), c(3,
            2, 3, 2, 3), c(2, 2, 1, 2, 2), c(2, 2, 3, 2, 2), c(1, 1, 3, 1, 1), c(3,
            3, 1, 3, 3), c(1, 3, 3, 3, 1), c(3, 1, 1, 1, 3), c(2, 3, 1, 3, 2), c(1,
            1, 2, 3, 2), c(3, 2, 1, 2, 1), c(1, 1, 1, 1, 2), c(3, 3, 3, 3, 2), c(2,
            2, 1, 1, 1), c(2, 2, 3, 3, 3)))
    }

    # add more paylines
    if (num_lines == 40) {
        payline_patterns <- append(payline_patterns, list(c(1, 2, 1, 2, 1), c(3,
            2, 3, 2, 3), c(2, 2, 1, 2, 2), c(2, 2, 3, 2, 2), c(1, 1, 3, 1, 1), c(3,
            3, 1, 3, 3), c(1, 3, 3, 3, 1), c(3, 1, 1, 1, 3), c(2, 3, 1, 3, 2), c(1,
            1, 2, 3, 2), c(3, 2, 1, 2, 1), c(1, 1, 1, 1, 2), c(3, 3, 3, 3, 2), c(2,
            2, 1, 1, 1), c(2, 2, 3, 3, 3), c(1, 2, 2, 2, 1), c(3, 2, 2, 2, 3), c(1,
            3, 2, 1, 2), c(2, 1, 2, 3, 3), c(3, 2, 1, 3, 1), c(1, 1, 1, 3, 3), c(3,
            3, 3, 1, 1), c(2, 2, 2, 1, 2), c(1, 3, 3, 3, 2), c(3, 1, 1, 1, 2)))
    }

    # define payouts for certain outcomes
    payouts <- list(`1` = list(three = 71, four = 181, five = 362), `2` = list(three = 71,
        four = 191, five = 372), `3` = list(three = 71, four = 201, five = 382),
        `4` = list(three = 71, four = 191, five = 372), `5` = list(three = 71, four = 181,
            five = 362), `6` = list(three = 71, four = 201, five = 382), `7` = list(three = 71,
            four = 191, five = 372), `8` = list(three = 71, four = 181, five = 362),
        `9` = list(three = 71, four = 201, five = 382), `10` = list(three = 71, four = 191,
            five = 372), `11` = list(three = 71, four = 181, five = 362), `12` = list(three = 71,
            four = 201, five = 382), `13` = list(three = 111, four = 241, five = 491),
        `14` = list(three = 111, four = 251, five = 501), `15` = list(three = 151,
            four = 331, five = 661))

    # iterate through list to check paylines for matching payouts
    for (pattern in payline_patterns) {
        line_symbols <- reels[cbind(pattern, 1:5)]
        symbol <- line_symbols[1]

        # replace wild symbol with symbol
        line_symbols[line_symbols == 15] <- symbol

        # skip if first symbol is blank
        if (symbol == 0)
            next

        # check for 3, 4, 5 of same symbol and pay corresponding payout
        if (line_symbols[2] == symbol & line_symbols[3] == symbol & line_symbols[4] ==
            symbol & line_symbols[5] == symbol) {
            total_wins <- total_wins + payouts[[as.character(symbol)]]$five * bet_size_per_line
            payline_wins <- payline_wins + 1
        } else if (line_symbols[2] == symbol & line_symbols[3] == symbol & line_symbols[4] ==
            symbol) {
            total_wins <- total_wins + payouts[[as.character(symbol)]]$four * bet_size_per_line
            payline_wins <- payline_wins + 1
        } else if (line_symbols[1] == symbol & line_symbols[2] == symbol & line_symbols[3] ==
            symbol) {
            total_wins <- total_wins + payouts[[as.character(symbol)]]$three * bet_size_per_line
            payline_wins <- payline_wins + 1
        }
    }

    total_bet <- num_lines * bet_size_per_line
    jackpot_env$progressive_jackpot <- jackpot_env$progressive_jackpot + 0.02 * total_bet

    # check if jackpot is hit
    if (jackpot_env$progressive_jackpot >= jackpot_env$target) {
        total_wins <- total_wins + jackpot_env$progressive_jackpot
        jackpot_env$progressive_jackpot <- 1000
        jackpot_env$target <- sample(1000:10000, 1)
    }

    # bonus feature if bonus symbols on reels 1, 3, 5
    if (any(reels[1:3, 1] == 13) & any(reels[1:3, 3] == 13) & any(reels[1:3, 5] ==
        13)) {
        bonus_payouts <- sample(100:500, 5)
        player_choice <- sample(1:5, 1)
        total_wins <- total_wins + bonus_payouts[player_choice]
    }

    # return total wins
    return(list(total_wins = total_wins, payline_wins = payline_wins))
}

five_reel_machine <- function(slot_players, number_machines = 100) {
    # assign player id
    for (i in seq_along(slot_players)) {
        slot_players[[i]]$id <- i
    }

    # split players across machines and cycle through active and waiting
    # players
    active_players <- slot_players[1:min(number_machines, length(slot_players))]
    waiting_players <- slot_players[-seq_len(length(active_players))]

    # cycle through every player and every round
    while (any(sapply(slot_players, function(x) x$games_played > 0))) {
        for (machine_index in seq_along(active_players)) {
            player <- active_players[[machine_index]]

            if (player$games_played > 0) {
                # randomly choose number of lines and bet size per line
                num_lines <- sample(c(15, 30, 40), 1)
                bet_size_per_line <- sample(10:30, 1)

                # use previous 5-reel function to get results
                result <- five_reel_slots(num_lines, bet_size_per_line)

                # update players
                player$result <- player$result + result$total_wins
                player$bet_amount <- player$bet_amount + (num_lines * bet_size_per_line)
                player$games_played <- player$games_played - 1
            }

            # ycle through players
            if (player$games_played <= 0 & length(waiting_players) > 0) {
                active_players[[machine_index]] <- waiting_players[[1]]
                waiting_players <- waiting_players[-1]
            } else {
                active_players[[machine_index]] <- player
            }

            # update lists
            slot_players[[player$id]] <- player
        }

        active_players <- Filter(function(p) !is.null(p) & p$games_played > 0, active_players)
    }

    # histogram of winnings
    winnings <- sapply(slot_players, function(x) x$result)
    hist(winnings, breaks = 50, col = "blue", border = "black", main = "Histogram of 5-Reel Slots Winnin
        xlab = "Winnings", ylab = "Frequency")

    # get total results and calculate house advantage
    total_result <- sum(sapply(slot_players, function(x) x$result))
    total_bet_amount <- sum(sapply(slot_players, function(x) x$bet_amount))
    house_advantage <- (total_bet_amount - total_result)/total_bet_amount

    return(list(total_result = total_result, total_bet_amount = total_bet_amount,
        house_advantage = house_advantage))
}

# split 5-reel players
five_reel_players <- players_split[["5-Reel"]]
```

```
# simulate to get results
five_reel_result <- five_reel_machine(five_reel_players)
```

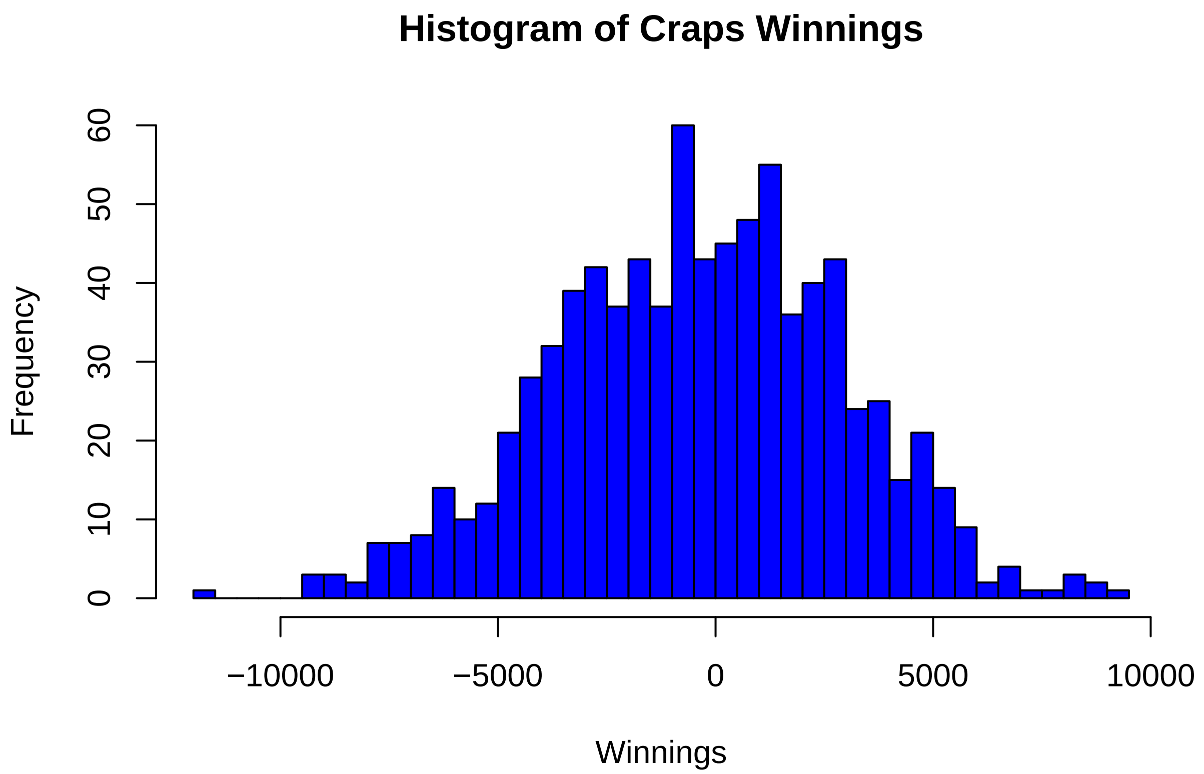


Craps:

```
# new function for craps to create craps tables same logic as original craps
# function with slightly different parameters

craps <- function(bet_type, bet_size, free_odds, casino_outcome = NULL) {

    # function to simulate dice roll
    diceroll <- function() {
        return(sum(sample(1:6, 2, replace = TRUE)))
    }

    # function to calculate free odds payouts
    free_odd_pay <- function(point) {
        if (point %in% c(4, 10))
            return(2/1)
        if (point %in% c(5, 9))
            return(3/2)
        if (point %in% c(6, 8))
            return(6/5)
    }

    # set initial variables
    point <- NULL
    payout <- 0
    odd_bet <- bet_size * free_odds

    # simulate casino come out roll
    if (is.null(casino_outcome)) {
        casino_outcome <- diceroll()
    }

    # logic to decide winner
    if (casino_outcome %in% c(7, 11)) {
        # immediate win for PASS, lose for DON'T PASS
        payout <- ifelse(bet_type == "PASS", bet_size, -bet_size)
    } else if (casino_outcome %in% c(2, 3)) {
        # immediate lose for PASS, win for DON'T PASS
        payout <- ifelse(bet_type == "PASS", -bet_size, bet_size)
    } else if (casino_outcome == 12) {
        # lose for PASS, tie for DON'T PASS
        payout <- ifelse(bet_type == "PASS", -bet_size, 0)
    } else {
        # establish point
        point <- casino_outcome

        repeat {
            roll <- diceroll()
            if (roll == point) {
                # point made
                if (bet_type == "PASS") {
                  payout <- bet_size + free_odd_pay(point) * odd_bet
                } else {
                  payout <- -bet_size - odd_bet * free_odd_pay(point)
                }
                break
            } else if (roll == 7) {

                if (bet_type == "PASS") {
                  payout <- -bet_size - odd_bet
                } else {
                  payout <- bet_size + odd_bet
                }
                break
            }
        }
    }

    # calculates payouts based on bet type
    return(payout)
}

# function to calculate results for craps table (3 tables with 8 players each)

craps_table <- function(craps_players, number_tables = 3, max_players = 8) {
    # assign player id
```

```
for (i in seq_along(craps_players)) {
    craps_players[[i]]$id <- i
}

# determine active and waiting players
active_players <- craps_players[1:min(number_tables * max_players, length(craps_players))]
waiting_players <- craps_players[-(1:length(active_players))]

# split active players into each table
table_indices <- rep(1:number_tables, length.out = length(active_players))
tables <- split(active_players, table_indices)

# loop until no players have any games remaining
while (any(sapply(craps_players, function(x) x$games_played > 0))) {
    for (table_index in seq_along(tables)) {
        table <- tables[[table_index]]

        for (i in seq_along(table)) {
            player <- table[[i]]
            if (player$games_played > 0) {
              # choose random bet type, bet size (10 to 100), and free odds
              # (1 to 3)
              bet_type <- sample(c("PASS", "DON'T PASS"), 1)
              bet_size <- sample(10:100, 1)
              free_odds <- sample(1:3, 1)

              # get craps result from previous function
              player_result <- craps(bet_type, bet_size, free_odds)

              # update player results
              player$result <- player$result + player_result
              player$bet_amount <- player$bet_amount + bet_size * (1 + free_odds)
              player$games_played <- player$games_played - 1

              # cycle through waiting players
              if (player$games_played <= 0 & length(waiting_players) > 0) {
                table[[i]] <- waiting_players[[1]]
                waiting_players <- waiting_players[-1]
              } else {
                table[[i]] <- player
              }

              # update lists
              player_id <- player$id
              craps_players[[player_id]] <- player
            }
        }
        tables[[table_index]] <- table
    }
}

# histogram of winnings
winnings <- sapply(craps_players, function(x) x$result)
```

```r
    hist(winnings, breaks = 50, col = "blue", border = "black", main = "Histogram of Craps Winnings",
        xlab = "Winnings", ylab = "Frequency")

    # calculate total results and house advantage
    total_result <- sum(sapply(craps_players, function(x) x$result))
    total_bet_amount <- sum(sapply(craps_players, function(x) x$bet_amount))
    house_advantage <- -(total_result/total_bet_amount)

    return(list(total_result = total_result, total_bet_amount = total_bet_amount,
        house_advantage = house_advantage))
}

# split players into craps players
craps_players <- players_split[["Craps"]]

craps_result <- craps_table(craps_players)
```



Summary of Results:

```r
# combine results into list
results <- list(Roulette = roulette_result, Baccarat = baccarat_result, Keno = keno_result,
    `4-Reel Slots` = four_reel_result, `5-Reel Slots` = five_reel_result, Craps = craps_result)

# create dataframe to store results
summary_df <- data.frame(Game = character(), Total_Bet = numeric(), Total_Won = numeric(),
    Casino_Profit = numeric(), House_Advantage = numeric(), stringsAsFactors = FALSE)
```

```r
# put results into dataframe
for (game in names(results)) {
    res <- results[[game]]

    # adjust for how game winnings are calculated for different games
    total_won <- if (game %in% c("Roulette", "Baccarat", "Craps")) {
        res$total_result + res$total_bet_amount
    } else {
        res$total_result
    }

    total_bet <- res$total_bet_amount
    casino_profit <- total_bet - total_won
    house_advantage <- casino_profit/total_bet

    # add data to dataframe
    summary_df <- rbind(summary_df, data.frame(Game = game, Total_Bet = total_bet,
        Total_Won = total_won, Casino_Profit = casino_profit, House_Advantage = house_advantage))
}

# add totals
total_bet_sum <- sum(summary_df$Total_Bet)
total_won_sum <- sum(summary_df$Total_Won)
casino_profit_sum <- sum(summary_df$Casino_Profit)
house_advantage_total <- casino_profit_sum/total_bet_sum

summary_df <- rbind(summary_df, data.frame(Game = "Total", Total_Bet = total_bet_sum,
    Total_Won = total_won_sum, Casino_Profit = casino_profit_sum, House_Advantage = house_advantage_tota

# print table using kable
library(knitr)
kable(summary_df, caption = "Casino Performance Summary", format = "simple", digits = 4)
```

Table 1: Casino Performance Summary

Game	Total_Bet	Total_Won	Casino_Profit	House_Advantage
Roulette	21210528	20134466	1076062.0	0.0507
Baccarat	3893373	3421828	471545.4	0.1211
Keno	743977	540207	203770.0	0.2739
4-Reel Slots	111718430	82910512	28807918.0	0.2579
5-Reel Slots	2445063130	2027056001	418007129.2	0.1710
Craps	45844829	45506854	337975.3	0.0074
Total	2628474267	2179569867	448904399.9	0.1708

Part 3:

Creating a successful players club would involve many considerations. I will outline a few important features that I have thought of when brainstorming. First, I would create a system that has different rewards for different types of games and bets. Higher stakes bets, greater time spent in the casino, and different game types would all contribute the the reward a player receives. Players who bet more on each hand, spend longer at the casino, and play higher house advantage games, such as slots, would recieve greater rewards, as they will earn more money for the casino. For the actual rewards, I would create a tiered system. Higher tiers would receive more expensive benefits, as they are the players betting more, for longer. Each tier would receive different rewards per bet amount. Maybe a bronze player would earn 1% of their bet amount in rewards, while a high-stakes gold player would receive 5% of their bet amount. This amount would have to be weighted differently for different bet types, games, and amounts, but these figures could be the baseline. Along with rewards money that players can bet with, we can offer players free comps. Bronze players would get smaller valued comps such as free drinks when they are playing. There isn't much risk in this for the casino. Gold players might get a free flight or a free night stay at the casino. Even though these would be higher risk for the casino, these high stakes players would most likely make the casino more money in bets, making it worth it. To see how the players program is working, we could observe key metrics. We could look at the retention rates for the players we are giving free rewards and comps to. We could also see if these programs are making enough profit for the casino for it to be worth it. A basic formula for rewards could be rewards_points = total_bets * game_factor * time_factor. This accounts for the bet amount, time spent in the casino, and games they are playing. These factors could be changed based on how the program works what is optimal for the casino, but in general, the rewards would increase, as players play higher house advantage games, spend longer times at the casino, and bet more per session.

```
  total_result <- sum(sapply(craps_players, function(x) x$result))
  total_bet_amount <- sum(sapply(craps_players, function(x) x$bet_amount))
  house_advantage <- -(total_result / total_bet_amount)

  return(list(
    total_result = total_result,
    total_bet_amount = total_bet_amount,
    house_advantage = house_advantage
  ))
}
```

Part 2:

```
library(knitr)
# function to set new seed after breaks
reset_rng <- function() {
    new_seed <- sample(1:1e+05, 1)
    set.seed(new_seed)
}

# initialize budget
initial_budget <- 4e+05

daily_play_time <- function() rgamma(1, 800, 2)  #generat total minutes per day
break_time <- function() rgamma(1, 10, 0.5)  #generate break duration
play_session_time <- function() rgamma(1, 40, 0.5)  #generate duration per session before break

game_probs <- c(0.1, 0.05, 0.2, 0.6, 0.05)

games_per_hour <- c(roulette = 50, baccarat = 50, keno = 10, four_reel = 400, five_reel = 400,
    craps = 100)

# function to simulate game with slightly updated parameters
roulette_game <- function(budget) {
    bet_size <- 500  #fixed bet
    if (budget >= bet_size) {
        # randomly samples bet type
        result <- roulette(sample(c("single_number", "red_black", "even_odd", "low_high",
            "columns", "dozens", "corner", "street"), 1), bet_size, sample(1:38,
            1))
        net_winnings <- result
        return(list(bet = bet_size, winnings = net_winnings))
    }
    return(list(bet = 0, winnings = 0))
}

# function to simulate game with slightly updated parameters
baccarat_game <- function(budget) {
    bet_size <- (rpois(1, 5) + 1) * 100  #bet sized specified in prompt
    if (budget >= bet_size) {

        # randomly selects bet type
        result <- baccarat(sample(c("banker", "player", "tie"), 1), bet_size, NULL)
        net_winnings <- result
        return(list(bet = bet_size, winnings = net_winnings))
    }
    return(list(bet = 0, winnings = 0))
}

# function to simulate game with slightly updated parameters
four_reel_game <- function(budget) {
    bet_multipliers <- c(1, 2, 3)
    bet_size <- sample(bet_multipliers, 1) * 3 * 100  #betting on all lines with a random bet size and
    if (budget >= bet_size) {
        result <- four_reel_slots(3, bet_size)
        total_winnings <- result
        return(list(bet = bet_size, winnings = total_winnings))
    }
    return(list(bet = 0, winnings = 0))
}

# function to simulate game with slightly updated parameters
five_reel_game <- function(budget) {
    bet_size_per_line <- sample(1:30, 1)  #random sampling 1-5 units per line
    lines <- sample(c(15, 30, 40), 1)  #randomly sampling amoung of lines
    total_bet <- bet_size_per_line * lines
    if (budget >= total_bet) {
        result <- five_reel_slots(lines, bet_size_per_line)
        total_winnings <- result$total_wins
        return(list(bet = total_bet, winnings = total_winnings))
    }
    return(list(bet = 0, winnings = 0))
}

# function to simulate game with slightly updated parameters
craps_game <- function(budget) {
    bet_size <- sample(c(100, 200, 300, 400, 500), 1)  #randomly picking bet size in increments of 100
    bet_type <- sample(c("PASS", "DON'T PASS"), 1)  #randomly choosing bet type
    free_odds <- sample(1:3, 1)
    if (budget >= bet_size) {
        result <- craps(bet_type, bet_size, free_odds)
        net_winnings <- result
        return(list(bet = bet_size, winnings = net_winnings))
    }
    return(list(bet = 0, winnings = 0))
}

# mapping games
game_functions <- list(roulette_game, baccarat_game, four_reel_game, five_reel_game,
    craps_game)

# function to simulate 1 single day
simulate_day <- function(player_budget) {
    total_minutes <- daily_play_time()
    time_remaining <- total_minutes
    total_bets <- 0
    total_won <- 0

    while (time_remaining > 0 && player_budget > 0) {
        session_time <- min(play_session_time(), time_remaining)
        time_remaining <- time_remaining - session_time

        # determine amount of rounds based on game type and time
        game_type <- sample(1:5, 1, prob = game_probs)
        games_per_hour <- c(50, 50, 400, 400, 100)[game_type]
        rounds <- floor((session_time/60) * games_per_hour)

        for (round in 1:rounds) {
            game_result <- game_functions[[game_type]](player_budget)

            # update winnings
            total_bets <- total_bets + game_result$bet
            total_won <- total_won + game_result$winnings
            player_budget <- player_budget - game_result$bet + game_result$winnings

            if (player_budget <= 0)
                break  #stop if out of money
        }

        # simulate break
        if (time_remaining > 0) {
            time_remaining <- time_remaining - break_time()
            reset_rng()  #reset seed
        }
    }

    return(list(total_bets = total_bets, total_won = total_won, remaining_budget = player_budget))
}

# simulate four days
simulate_four_days <- function(initial_budget) {
    day_results <- list()
    current_budget <- initial_budget

    for (day in 1:4) {
        day_result <- simulate_day(current_budget)
        current_budget <- day_result$remaining_budget
        day_results[[day]] <- day_result
    }

    # summarize results
    total_bets <- sum(sapply(day_results, function(x) x$total_bets))
    total_won <- sum(sapply(day_results, function(x) x$total_won))
    final_budget <- current_budget
    house_advantage <- (total_bets - total_won)/total_bets

    return(list(day_results = day_results, total_bets = total_bets, total_won = total_won,
        final_budget = final_budget, house_advantage = house_advantage))
}
```

```
# run simulation
results <- simulate_four_days(initial_budget)

# multiplying house advantages from simulations by game probability
theoretical_house_advantage <- 0.0507 * 0.1 + 0.1211 * 0.05 + 0.2579 * 0.2 + 0.171 *
    0.6 + 0.0074 * 0.05

day_results <- do.call(rbind, lapply(1:4, function(day) {
    total_bets <- results$day_results[[day]]$total_bets
    total_won <- results$day_results[[day]]$total_won
    house_advantage <- (total_bets - total_won)/total_bets
    list(Day = day, Total_Bets = total_bets, Total_Won = total_won, Remaining_Budget = results$day_resul
        House_Advantage = round(house_advantage * 100, 2), Theoretical_House_Advantage = round(theoreti
            100, 2))
}))

summary_row <- data.frame(Day = "Summary", Total_Bets = results$total_bets, Total_Won = results$total_w
    Remaining_Budget = results$final_budget, House_Advantage = round(results$house_advantage *
        100, 2), Theoretical_House_Advantage = round(theoretical_house_advantage *
        100, 2))

formatted_results <- rbind(data.frame(day_results, stringsAsFactors = FALSE), summary_row)

# prints results nicely in a table using kable
kable(formatted_results, col.names = c("Day", "Total Bets", "Total Won", "Remaining Budget",
    "House Advantage (%)", "Theoretical House Advantage (%)"), caption = "Simulation Results for 4 Days
```

Table 1: Simulation Results for 4 Days

Day	Total Bets	Total Won	Remaining Budget	House Advantage (%)	Theoretical House Advantage (%)
1	523095	358519	235424	31.46	16.57
2	534570	458661	159515	14.2	16.57
3	573845	414341.3	11.3	27.8	16.57
4	0	0	11.3	NaN	16.57
Summary	1631510	1231521	11.3	24.52	16.57