Simulation Results

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

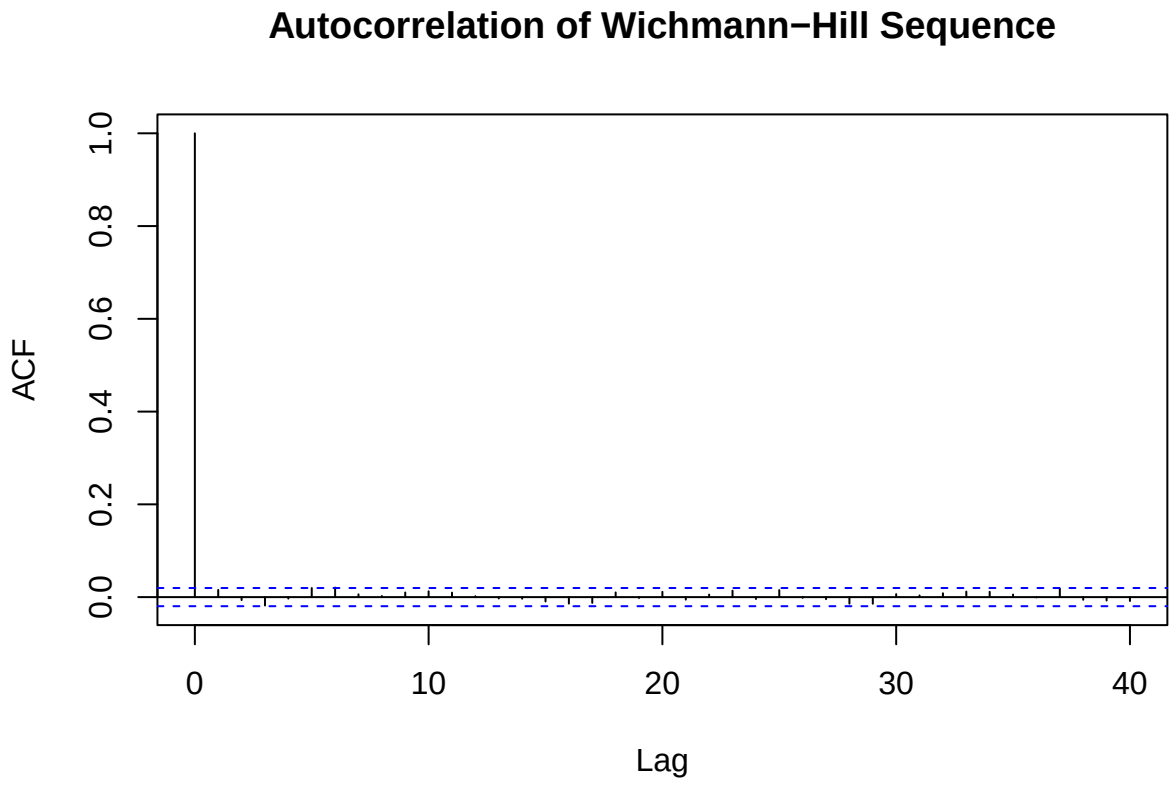
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
rng.chisq <- function(x, m=10) {
  Ob <- trunc(m * x) / m
  Ob <- table(Ob)
  p <- rep(1, m) / m
  Ex <- length(x) * p
  chisq <- sum((Ob - Ex)^2 / Ex)
  pvalue <- 1 - pchisq(chisq, m - 1)
  list(test.statistic = chisq, p.value = pvalue, df = m - 1)
}

perm.test <- function(u, k = 3) {
  if ((length(u) %% k) != 0) warning("Input sequence length is not a multiple of k")
  u <- matrix(u, nrow = k)  # split into n subsequences
  u <- data.frame(u)
  Obs <- table(sapply(u, function(x) paste(order(x), collapse = "")))
  n <- ncol(u)
  Exp <- n / factorial(k)
  x <- sum((Obs - Exp)^2 / Exp)
  pvalue <- 1 - pchisq(x, factorial(k) - 1)
  list(test.statistic = x, p.value = pvalue, Observed = Obs, Expected = Exp)
}

wichmann.hill <- function(n, seed = c(1, 1, 1)) {
  s1 <- seed[1]
  s2 <- seed[2]
  s3 <- seed[3]
  u <- numeric(n)
  for (i in 1:n) {
    s1 <- (171 * s1) %% 30269
    s2 <- (172 * s2) %% 30307
    s3 <- (170 * s3) %% 30323
    u[i] <- (s1 / 30269 + s2 / 30307 + s3 / 30323) %% 1
  }
  u
}

set.seed(123)
x <- wichmann.hill(10000)
```

```r
uniform_test <- rng.chisq(x, m = 10)
print(paste("result for uniformatiy:", uniform_test))
```

```
## [1] "result for uniformatiy: 8.528"
## [2] "result for uniformatiy: 0.481932232430112"
## [3] "result for uniformatiy: 9"
```

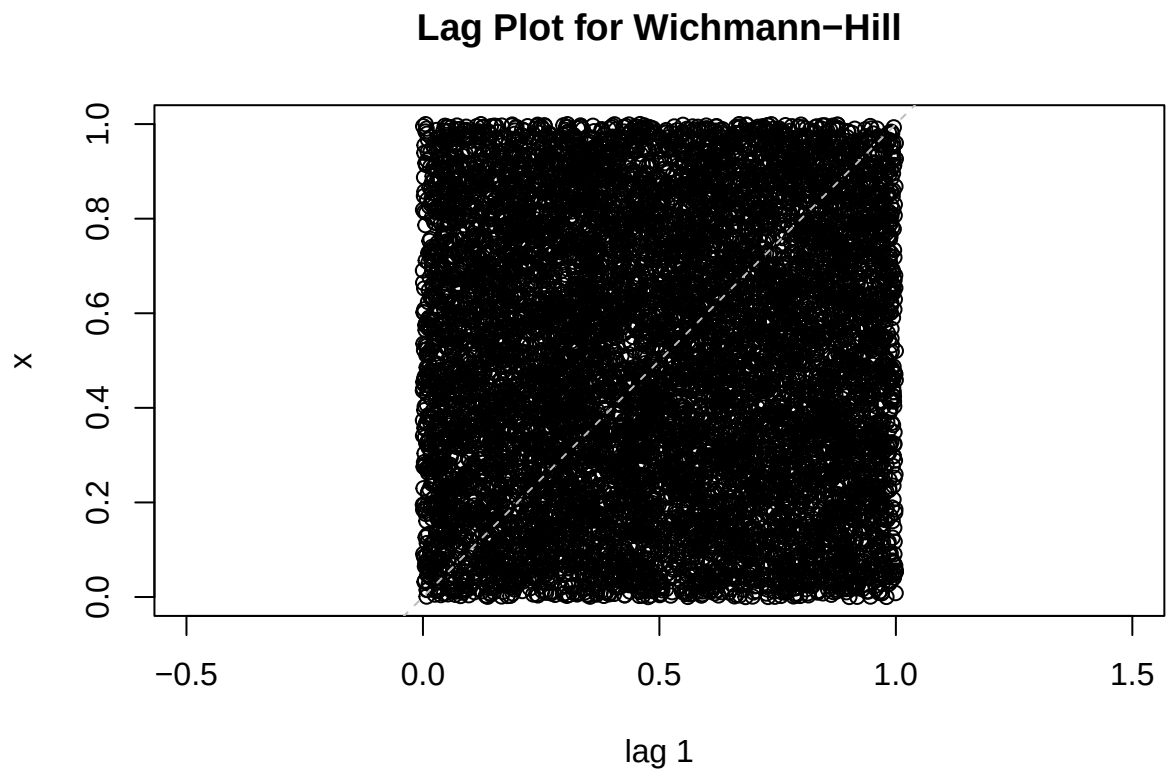
```r
acf(x, main = "Autocorrelation of Wichmann-Hill Sequence")
```




```r
k_values <- c(4, 5)
perm_results <- list()
for (k in k_values) {
  if (10000 %% k != 0) {
    warning(paste("10000 not multiple of", k, "- trimming data"))
    u_trim <- x[1:(10000 - (10000 %% k))]
  } else {
    u_trim <- x
  }
  perm_results[[as.character(k)]] <- perm.test(u_trim, k = k)
}
print(paste("permutation result:", perm_results))
```

```
## [1] "permutation result: list(test.statistic = 19.424, p.value = 0.676359604980626, Observed = c(‘123
## [2] "permutation result: list(test.statistic = 100.24, p.value = 0.893013153317069, Observed = c(‘123
```

```r
lag.plot(x, lags = 1, do.lines = FALSE, main = "Lag Plot for Wichmann-Hill")
```



#Q2

```r
set.seed(123)
n <- 10000
z <- runif(n, min = 3.7, max = 5.8)

mean_est <- mean(z)
var_est <- var(z)
sd_est <- sd(z)


mean_true <- (3.7 + 5.8) / 2  # 4.75
var_true <- (5.8 - 3.7)^2 / 12  # 0.3675
sd_true <- sqrt(var_true)  # 0.6063

cat("Estimated Mean:", mean_est, "vs True Mean:", mean_true, "\n")
cat("Estimated Variance:", var_est, "vs True Variance:", var_true, "\n")
cat("Estimated Standard Deviation:", sd_est, "vs True Standard Deviation:", sd_true, "\n")

prob_est <- mean(z > 4.0)

prob_true <- (5.8 - 4.0) / (5.8 - 3.7)  # 0.8571
```

```
cat("Estimated Probability P(z > 4.0):", prob_est, "\n")
cat("True Probability P(z > 4.0):", prob_true, "\n")

#Q3

install.packages("shiny")
install.packages("grid")

library(shiny)
library(grid)

#####################################################
propertycolors <- c("grey", "purple", "grey", "purple", "grey",
                    "grey", "lightblue", "grey", "lightblue", "lightblue", "black", "magenta",
                    "grey", "magenta", "magenta", "grey", "orange", "grey", "orange",
                    "orange", "grey", "red", "grey", "red", "red", "grey", "yellow",
                    "yellow", "grey", "yellow", "black", "green", "green",
                    "grey", "green", "grey", "grey", "darkblue", "grey", "darkblue")


ui <- shinyUI(pageWithSidebar(
  headerPanel("Monopoly Game - Two Players"),

  sidebarPanel(
    sliderInput("nplay1", "Number of plays for Player 1",
                0, 1000, 0, step = 1,
                animate=animationOptions(interval=1200, loop=T,
                                         playButton=HTML("Play/Pause"),
                                         pauseButton=HTML("Play/Pause"))),
    numericInput('nplay1_manual', 'Or type Player 1 plays (more than 1000):', 0),
    br(),
    sliderInput("nplay2", "Number of plays for Player 2",
                0, 1000, 0, step = 1,
                animate=animationOptions(interval=1200, loop=T,
                                         playButton=HTML("Play/Pause"),
                                         pauseButton=HTML("Play/Pause"))),
    numericInput('nplay2_manual', 'Or type Player 2 plays (more than 1000):', 0),
    br(),
    helpText("Note: The game board shows the last outcome with the sum of two dice.",
             "Histograms reflect landing patterns for both players."),
    br(),
    div(actionButton('con', 'Continue')),
    br(),
    div(actionButton('so', 'Start over')),
    tags$hr(),
    radioButtons("type", "Choose histogram type:",
                 list("Frequency" = "cou",
                      "Percentage" = "per"))
  ),

  mainPanel(
    tabsetPanel(
      tabPanel("Monopoly Game", plotOutput("main_plot"),
               br(), downloadButton('png1', 'Printer-friendly Version')),
```

```
      tabPanel("Bar Chart - Player 1", plotOutput("hist1"),
               br(), downloadButton('png2', 'Printer-friendly Version')),
      tabPanel("Bar Chart - Player 2", plotOutput("hist2"),
               br(), downloadButton('png3', 'Printer-friendly Version')),
      tabPanel('Locations',
               downloadButton('tab0', 'Printer-friendly Version'),
               br(), tableOutput("check"))
    )
  )
))

server <- shinyServer(function(input, output, session) {

  seed0 <- rnorm(1, 100, 10000)
  ########################################
  observe({
    if(input$so == 0) return(NULL)
    isolate({
      updateSliderInput(session, 'nplay1',
                        "Number of plays for Player 1 or click Play/Pause to watch an animation", 0)
      updateSliderInput(session, 'nplay2',
                        "Number of plays for Player 2 or click Play/Pause to watch an animation", 0)
    })
  })

  observe({
    if(input$con == 0) return(NULL)
    isolate({
      updateSliderInput(session, 'nplay1',
                        "Number of plays for Player 1 or click Play/Pause to watch an animation", input$
      updateSliderInput(session, 'nplay2',
                        "Number of plays for Player 2 or click Play/Pause to watch an animation", input$
    })
  })

  observe({
    if(input$nplay1 > 1000) return(NULL)
    isolate({
      updateNumericInput(session, 'nplay1_manual', 'Or type Player 1 plays (more than 1000):', input$npl
    })
  })

  observe({
    if(input$nplay2 > 1000) return(NULL)
    isolate({
      updateNumericInput(session, 'nplay2_manual', 'Or type Player 2 plays (more than 1000):', input$npl
    })
  })

  observe({
    if(is.na(input$nplay1_manual) || input$nplay1_manual > 1000) return(NULL)
    isolate({
      updateSliderInput(session, 'nplay1',
                        "Number of plays for Player 1 or click Play/Pause to watch an animation", input$
```

```
  })
})

observe({
  if(is.na(input$nplay2_manual) || input$nplay2_manual > 1000) return(NULL)
  isolate({
    updateSliderInput(session, 'nplay2',
                      "Number of plays for Player 2 or click Play/Pause to watch an animation", input$
  })
})
########################################
land1 <- reactive({
  seed <- seed0 + (input$so)
  if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
  lands <- numeric(n + 1)
  lands[1] <- 1
  currentthrow <- c(0, 1)
  if(n > 0) {
    set.seed(seed * 1) # Unique seed for Player 1
    throw <- sample(1:6, size = n * 2, replace = TRUE)
    currentthrow <- throw[c(2 * n - 1, 2 * n)]
    for(i in 2:(n + 1)) {
      lands[i] <- (lands[i - 1] + sum(throw[c((i - 2) * 2 + 1, (i - 1) * 2)]) - 1) %% 40 + 1
      if(lands[i] == 31) lands[i] <- 11
    }
  }
  c(lands, currentthrow)
})

land2 <- reactive({
  seed <- seed0 + (input$so + 1) # Different seed for Player 2
  if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
  lands <- numeric(n + 1)
  lands[1] <- 1
  currentthrow <- c(0, 1)
  if(n > 0) {
    set.seed(seed * 2) # Unique seed for Player 2
    throw <- sample(1:6, size = n * 2, replace = TRUE)
    currentthrow <- throw[c(2 * n - 1, 2 * n)]
    for(i in 2:(n + 1)) {
      lands[i] <- (lands[i - 1] + sum(throw[c((i - 2) * 2 + 1, (i - 1) * 2)]) - 1) %% 40 + 1
      if(lands[i] == 31) lands[i] <- 11
    }
  }
  c(lands, currentthrow)
})

# Generate histograms
output$hist1 <- renderPlot({
  if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
  lands <- land1()[1:(n + 1)]
  if(input$type == 'cou') {
    ylab <- "Number of Visits"
    flag <- FALSE
```

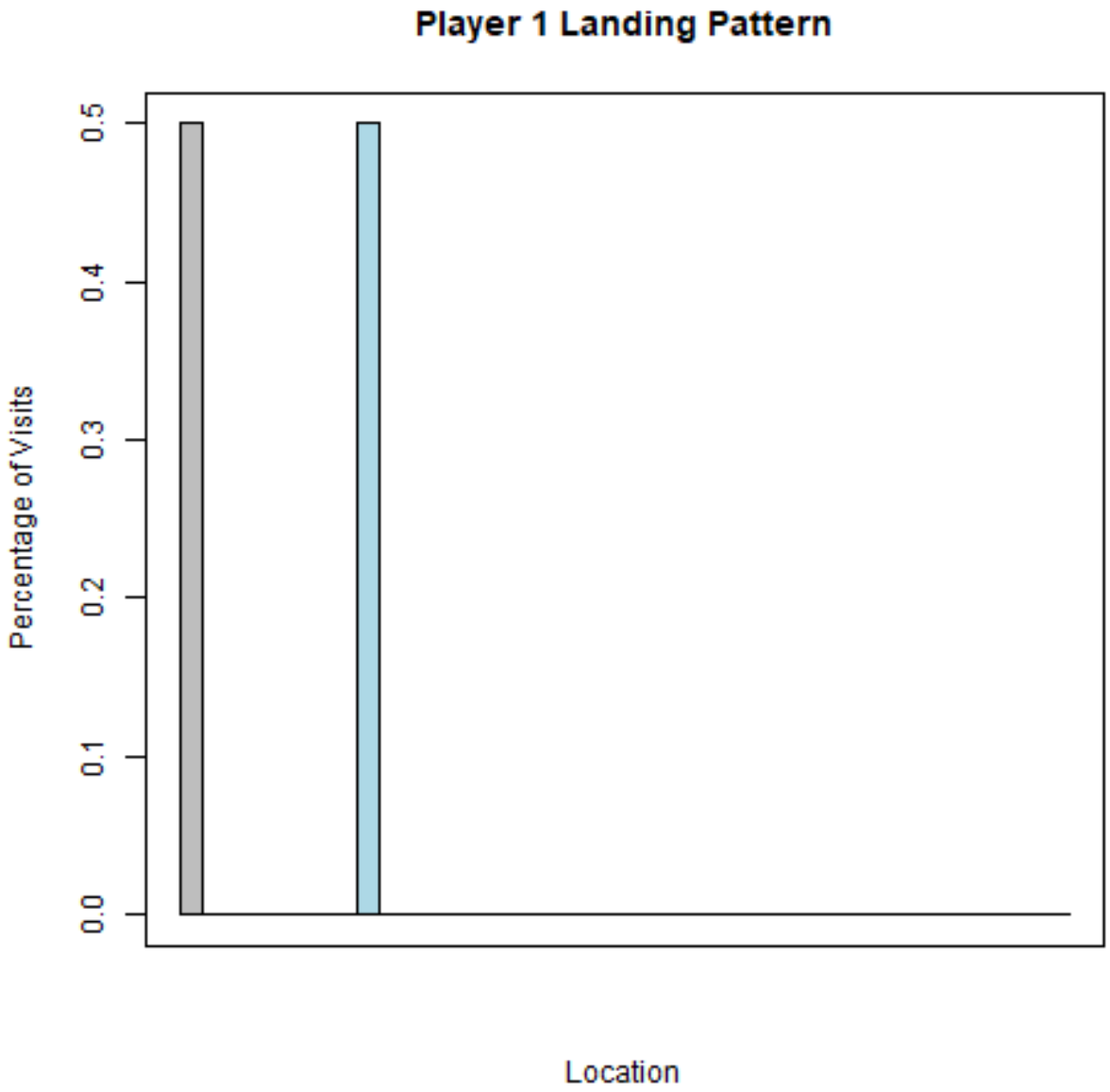
```
  } else {
    ylab <- 'Percentage of Visits'
    flag <- TRUE
  }
  hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
       xlab = "Location", main = "Player 1 Landing Pattern", ylab = ylab, axes = FALSE)
  box()
  axis(2)
})

output$hist2 <- renderPlot({
  if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
  lands <- land2()[1:(n + 1)]
  if(input$type == 'cou') {
    ylab <- "Number of Visits"
    flag <- FALSE
  } else {
    ylab <- 'Percentage of Visits'
    flag <- TRUE
  }
  hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
       xlab = "Location", main = "Player 2 Landing Pattern", ylab = ylab, axes = FALSE)
  box()
  axis(2)
})

output$png2 <- downloadHandler(
  filename = 'hist_Player1_Mono.png',
  content = function(file) {
    png(file)
    if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
    lands <- land1()[1:(n + 1)]
    if(input$type == 'cou') {
      ylab <- "Number of Visits"
      flag <- FALSE
    } else {
      ylab <- 'Percentage of Visits'
      flag <- TRUE
    }
    hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
         xlab = "Location", main = "Player 1 Landing Pattern", ylab = ylab, axes = FALSE)
    box()
    axis(2)
    dev.off()
  }
)

output$png3 <- downloadHandler(
  filename = 'hist_Player2_Mono.png',
  content = function(file) {
    png(file)
    if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
    lands <- land2()[1:(n + 1)]
    if(input$type == 'cou') {
```

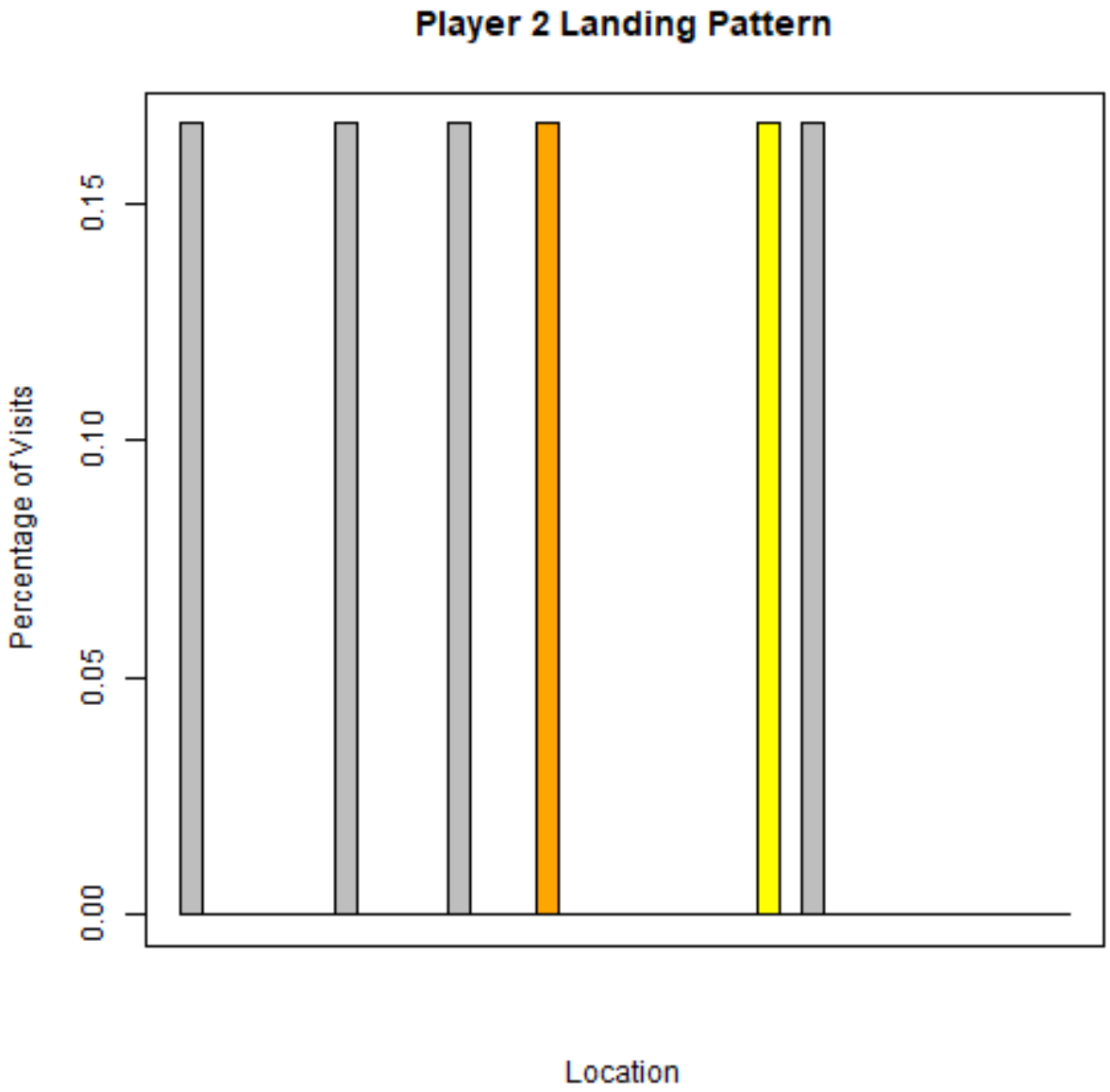
```
      ylab <- "Number of Visits"
      flag <- FALSE
    } else {
      ylab <- 'Percentage of Visits'
      flag <- TRUE
    }
    hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
         xlab = "Location", main = "Player 2 Landing Pattern", ylab = ylab, axes = FALSE)
    box()
    axis(2)
    dev.off()
  }
)
########################################
tab1 <- reactive({
  if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
  lands <- land1()[1:(n + 1)]
  df <- data.frame(lands)
  names(df) <- 'Location'
  df
})

tab2 <- reactive({
  if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
  lands <- land2()[1:(n + 1)]
  df <- data.frame(lands)
  names(df) <- 'Location'
  df
})

output$check <- renderTable({
  rbind(tab1(), tab2())
})

output$tab0 <- downloadHandler(
  filename = 'table_Mono.csv',
  content = function(file) {
    write.csv(rbind(tab1(), tab2()), file)
  }
)

########################################
output$main_plot <- renderPlot({
  grid.newpage()
  heights <- widths <- c(1.5, rep(1,9), 1.5)/12
  ycenter <- cumsum(c(1.5, rep(1,9), 1.5)/12) - widths/2
  xcenter <- rep(.75/12, 11)
  ypropcenter <- xpropcenter <- numeric(40)

  for (i in 1:11) {
    xpropcenter[i] <- xcenter[i]
    ypropcenter[i] <- ycenter[i]
    vp <- viewport(x = xpropcenter[i], y = ypropcenter[i],
                   height = heights[i], width = 1.5/12)
```

```
  pushViewport(vp)
  grid.rect(gp = gpar(fill = propertycolors[i]))
  upViewport()
}
########################################
vp <- viewport(x = xpropcenter[1], y = ypropcenter[1],
               height = heights[1], width = 1.5/12)
pushViewport(vp)
grid.text("Start", gp = gpar(col = "white", cex = 3))
upViewport()
########################################

for (i in 2:10) {
  xpropcenter[i + 10] <- ycenter[i]
  ypropcenter[i + 10] <- 1 - xcenter[i]
  vp <- viewport(x = xpropcenter[i + 10], y = ypropcenter[i + 10],
                 height = 1.5/12, width = widths[i])
  pushViewport(vp)
  grid.rect(gp = gpar(fill = propertycolors[10 + i]))
  upViewport()
}

for (i in 1:11) {
  xpropcenter[i + 20] <- 1 - xcenter[i]
  ypropcenter[i + 20] <- rev(ycenter)[i]
  vp <- viewport(x = xpropcenter[i + 20], y = ypropcenter[i + 20],
                 height = heights[i], width = 1.5/12)
  pushViewport(vp)
  grid.rect(gp = gpar(fill = (propertycolors[20 + i])))
  upViewport()
}

for (i in 2:10) {
  xpropcenter[i + 30] <- rev(ycenter)[i]
  ypropcenter[i + 30] <- xcenter[i]
  vp <- viewport(x = xpropcenter[i + 30], y = ypropcenter[i + 30],
                 height = 1.5/12, width = widths[i])
  pushViewport(vp)
  grid.rect(gp = gpar(fill = propertycolors[i + 30]))
  upViewport()
}

############################################
player <- circleGrob(r = .025, gp = gpar(fill = "white"), name = "player")
playerland <- editGrob(player, vp = viewport(x = xpropcenter[1],
                             y = ypropcenter[1]), name = "playerland")
dicevalue <- textGrob(x = .5, y = .5, "1", name = "dicevalue",
                      gp = gpar(col = 4, cex = 3))

grid.draw(dicevalue)
grid.draw(playerland)

if(is.na(input$nplay1) || input$nplay1 == 0) n1 <- 0 else n1 <- input$nplay1
if(is.na(input$nplay2) || input$nplay2 == 0) n2 <- 0 else n2 <- input$nplay2
```

```
  currentland1 <- land1()[n1 + 1]
  currentland2 <- land2()[n2 + 1]
  currentthrow1 <- land1()[c(n1 + 2, n1 + 3)]
  currentthrow2 <- land2()[c(n2 + 2, n2 + 3)]
  grid.edit("playerland", vp = viewport(x = xpropcenter[currentland1],
                                        y = ypropcenter[currentland1]))
  grid.edit("dicevalue", label = sum(currentthrow1), gp = gpar(col = 4, cex = 3))
  # Add Player 2 marker (e.g., red circle)
  player2land <- editGrob(player, gp = gpar(fill = "red"), vp = viewport(x = xpropcenter[currentland2]
                                                                         y = ypropcenter[currentland2]
  grid.draw(player2land)
})

output$png1 <- downloadHandler(
  filename = 'MonopolyGameBoard.png',
  content = function(file) {
    png(file)
    grid.newpage()
    heights <- widths <- c(1.5, rep(1,9), 1.5)/12
    ycenter <- cumsum(c(1.5, rep(1,9), 1.5)/12) - widths/2
    xcenter <- rep(.75/12, 11)
    ypropcenter <- xpropcenter <- numeric(40)

    for (i in 1:11) {
      xpropcenter[i] <- xcenter[i]
      ypropcenter[i] <- ycenter[i]
      vp <- viewport(x = xpropcenter[i], y = ypropcenter[i],
                     height = heights[i], width = 1.5/12)
      pushViewport(vp)
      grid.rect(gp = gpar(fill = propertycolors[i]))
      upViewport()
    }
    ########################################
    vp <- viewport(x = xpropcenter[1], y = ypropcenter[1],
                   height = heights[1], width = 1.5/12)
    pushViewport(vp)
    grid.text("Start", gp = gpar(col = "white", cex = 3))
    upViewport()
    ########################################

    for (i in 2:10) {
      xpropcenter[i + 10] <- ycenter[i]
      ypropcenter[i + 10] <- 1 - xcenter[i]
      vp <- viewport(x = xpropcenter[i + 10], y = ypropcenter[i + 10],
                     height = 1.5/12, width = widths[i])
      pushViewport(vp)
      grid.rect(gp = gpar(fill = propertycolors[10 + i]))
      upViewport()
    }

    for (i in 1:11) {
      xpropcenter[i + 20] <- 1 - xcenter[i]
      ypropcenter[i + 20] <- rev(ycenter)[i]
      vp <- viewport(x = xpropcenter[i + 20], y = ypropcenter[i + 20],
```

```
                    height = heights[i], width = 1.5/12)
      pushViewport(vp)
      grid.rect(gp = gpar(fill = (propertycolors[20 + i])))
      upViewport()
    }

    for (i in 2:10) {
      xpropcenter[i + 30] <- rev(ycenter)[i]
      ypropcenter[i + 30] <- xcenter[i]
      vp <- viewport(x = xpropcenter[i + 30], y = ypropcenter[i + 30],
                    height = 1.5/12, width = widths[i])
      pushViewport(vp)
      grid.rect(gp = gpar(fill = propertycolors[i + 30]))
      upViewport()
    }

    ################################################
    player <- circleGrob(r = .025, gp = gpar(fill = "white"), name = "player")
    playerland <- editGrob(player, vp = viewport(x = xpropcenter[1],
                                                 y = ypropcenter[1]), name = "playerland")
    dicevalue <- textGrob(x = .5, y = .5, "1", name = "dicevalue",
                          gp = gpar(col = 4, cex = 3))

    grid.draw(dicevalue)
    grid.draw(playerland)

    if(is.na(input$nplay1) || input$nplay1 == 0) n1 <- 0 else n1 <- input$nplay1
    if(is.na(input$nplay2) || input$nplay2 == 0) n2 <- 0 else n2 <- input$nplay2
    currentland1 <- land1()[n1 + 1]
    currentland2 <- land2()[n2 + 1]
    currentthrow1 <- land1()[c(n1 + 2, n1 + 3)]
    currentthrow2 <- land2()[c(n2 + 2, n2 + 3)]
    grid.edit("playerland", vp = viewport(x = xpropcenter[currentland1],
                                          y = ypropcenter[currentland1]))
    grid.edit("dicevalue", label = sum(currentthrow1), gp = gpar(col = 4, cex = 3))
    player2land <- editGrob(player, gp = gpar(fill = "red"), vp = viewport(x = xpropcenter[currentlan
                                                                           y = ypropcenter[currentlan
    grid.draw(player2land)
    dev.off()
  }
 )

})

shinyApp(ui = ui, server = server)

#The histograms for the two players are likely to show some differences due to the random nature of dic
```


Player 1 Landing Pattern
Percentage of Visits
0.0
0.1
0.2
0.3
0.4
0.5
Location



#Q4

```
library(shiny)
library(grid)

#######################################################
propertycolors <- c("grey", "purple", "grey", "purple", "grey",
                    "grey", "lightblue", "grey", "lightblue", "lightblue", "black", "magenta",
                    "grey", "magenta", "magenta", "grey", "orange", "grey", "orange",
                    "orange", "grey", "red", "grey", "red", "red", "grey", "yellow",
                    "yellow", "grey", "yellow", "black", "green", "green",
                    "grey", "green", "grey", "grey", "darkblue", "grey", "darkblue")

# Define UI
ui <- shinyUI(pageWithSidebar(
  headerPanel("Monopoly Game - Two Players with Revenue Analysis"),
```

```
  sidebarPanel(
    sliderInput("nplay1", "Number of plays for Player 1",
                0, 1000, 0, step = 1,
                animate=animationOptions(interval=1200, loop=T,
                                         playButton=HTML("Play/Pause"),
                                         pauseButton=HTML("Play/Pause"))),
    numericInput('nplay1_manual', 'Or type Player 1 plays (more than 1000):', 0),
    br(),
    sliderInput("nplay2", "Number of plays for Player 2",
                0, 1000, 0, step = 1,
                animate=animationOptions(interval=1200, loop=T,
                                         playButton=HTML("Play/Pause"),
                                         pauseButton=HTML("Play/Pause"))),
    numericInput('nplay2_manual', 'Or type Player 2 plays (more than 1000):', 0),
    br(),
    helpText("Note: The game board shows the last outcome with the sum of two dice.",
             "Histograms reflect landing patterns for both players."),
    br(),
    div(actionButton('con', 'Continue')),
    br(),
    div(actionButton('so', 'Start over')),
    div(actionButton('analyze', 'Run Revenue Analysis (10M Runs)')),
    tags$hr(),
    radioButtons("type", "Choose histogram type:",
                 list("Frequency" = "cou",
                      "Percentage" = "per"))
  ),

  mainPanel(
    tabsetPanel(
      tabPanel("Monopoly Game", plotOutput("main_plot"),
               br(), downloadButton('png1', 'Printer-friendly Version')),
      tabPanel("Bar Chart - Player 1", plotOutput("hist1"),
               br(), downloadButton('png2', 'Printer-friendly Version')),
      tabPanel("Bar Chart - Player 2", plotOutput("hist2"),
               br(), downloadButton('png3', 'Printer-friendly Version')),
      tabPanel("Revenue Analysis", verbatimTextOutput("revenue_output")),
      tabPanel('Locations',
               downloadButton('tab0', 'Printer-friendly Version'),
               br(), tableOutput("check"))
    )
  )
))

server <- shinyServer(function(input, output, session) {

  seed0 <- rnorm(1, 100, 10000)
  ########################################
  observe({
    if(input$so == 0) return(NULL)
    isolate({
      updateSliderInput(session, 'nplay1',
                        "Number of plays for Player 1 or click Play/Pause to watch an animation", 0)
```

```
    updateSliderInput(session, 'nplay2',
                      "Number of plays for Player 2 or click Play/Pause to watch an animation", 0)
  })
})

observe({
  if(input$con == 0) return(NULL)
  isolate({
    updateSliderInput(session, 'nplay1',
                      "Number of plays for Player 1 or click Play/Pause to watch an animation", input$
    updateSliderInput(session, 'nplay2',
                      "Number of plays for Player 2 or click Play/Pause to watch an animation", input$
  })
})

observe({
  if(input$nplay1 > 1000) return(NULL)
  isolate({
    updateNumericInput(session, 'nplay1_manual', 'Or type Player 1 plays (more than 1000):', input$npl
  })
})

observe({
  if(input$nplay2 > 1000) return(NULL)
  isolate({
    updateNumericInput(session, 'nplay2_manual', 'Or type Player 2 plays (more than 1000):', input$npl
  })
})

observe({
  if(is.na(input$nplay1_manual) || input$nplay1_manual > 1000) return(NULL)
  isolate({
    updateSliderInput(session, 'nplay1',
                      "Number of plays for Player 1 or click Play/Pause to watch an animation", input$
  })
})

observe({
  if(is.na(input$nplay2_manual) || input$nplay2_manual > 1000) return(NULL)
  isolate({
    updateSliderInput(session, 'nplay2',
                      "Number of plays for Player 2 or click Play/Pause to watch an animation", input$
  })
})
########################################
land1 <- reactive({
  seed <- seed0 + (input$so)
  if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
  lands <- numeric(n + 1)
  lands[1] <- 1
  currentthrow <- c(0, 1)
  if(n > 0) {
    set.seed(seed * 1)
    throw <- sample(1:6, size = n * 2, replace = TRUE)
```

```
    currentthrow <- throw[c(2 * n - 1, 2 * n)]
    for(i in 2:(n + 1)) {
      lands[i] <- (lands[i - 1] + sum(throw[c((i - 2) * 2 + 1, (i - 1) * 2)]) - 1) %% 40 + 1
      if(lands[i] == 31) lands[i] <- 11
    }
  }
  c(lands, currentthrow)
})

land2 <- reactive({
  seed <- seed0 + (input$so + 1)
  if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
  lands <- numeric(n + 1)
  lands[1] <- 1
  currentthrow <- c(0, 1)
  if(n > 0) {
    set.seed(seed * 2)
    throw <- sample(1:6, size = n * 2, replace = TRUE)
    currentthrow <- throw[c(2 * n - 1, 2 * n)]
    for(i in 2:(n + 1)) {
      lands[i] <- (lands[i - 1] + sum(throw[c((i - 2) * 2 + 1, (i - 1) * 2)]) - 1) %% 40 + 1
      if(lands[i] == 31) lands[i] <- 11
    }
  }
  c(lands, currentthrow)
})


output$hist1 <- renderPlot({
  if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
  lands <- land1()[1:(n + 1)]
  if(input$type == 'cou') {
    ylab <- "Number of Visits"
    flag <- FALSE
  } else {
    ylab <- 'Percentage of Visits'
    flag <- TRUE
  }
  hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
       xlab = "Location", main = "Player 1 Landing Pattern", ylab = ylab, axes = FALSE)
  box()
  axis(2)
})

output$hist2 <- renderPlot({
  if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
  lands <- land2()[1:(n + 1)]
  if(input$type == 'cou') {
    ylab <- "Number of Visits"
    flag <- FALSE
  } else {
    ylab <- 'Percentage of Visits'
    flag <- TRUE
  }
```

```
  hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
       xlab = "Location", main = "Player 2 Landing Pattern", ylab = ylab, axes = FALSE)
  box()
  axis(2)
})

output$png2 <- downloadHandler(
  filename = 'hist_Player1_Mono.png',
  content = function(file) {
    png(file)
    if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
    lands <- land1()[1:(n + 1)]
    if(input$type == 'cou') {
      ylab <- "Number of Visits"
      flag <- FALSE
    } else {
      ylab <- 'Percentage of Visits'
      flag <- TRUE
    }
    hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
         xlab = "Location", main = "Player 1 Landing Pattern", ylab = ylab, axes = FALSE)
    box()
    axis(2)
    dev.off()
  }
)

output$png3 <- downloadHandler(
  filename = 'hist_Player2_Mono.png',
  content = function(file) {
    png(file)
    if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
    lands <- land2()[1:(n + 1)]
    if(input$type == 'cou') {
      ylab <- "Number of Visits"
      flag <- FALSE
    } else {
      ylab <- 'Percentage of Visits'
      flag <- TRUE
    }
    hist(lands, breaks = seq(.5, 40.5, 1), probability = flag, col = propertycolors,
         xlab = "Location", main = "Player 2 Landing Pattern", ylab = ylab, axes = FALSE)
    box()
    axis(2)
    dev.off()
  }
)
########################################
tab1 <- reactive({
  if(is.na(input$nplay1) || input$nplay1 == 0) n <- 0 else n <- input$nplay1
  lands <- land1()[1:(n + 1)]
  df <- data.frame(lands)
  names(df) <- 'Location'
  df
```

```
})

tab2 <- reactive({
  if(is.na(input$nplay2) || input$nplay2 == 0) n <- 0 else n <- input$nplay2
  lands <- land2()[1:(n + 1)]
  df <- data.frame(lands)
  names(df) <- 'Location'
  df
})

output$check <- renderTable({
  rbind(tab1(), tab2())
})

output$tab0 <- downloadHandler(
  filename = 'table_Mono.csv',
  content = function(file) {
    write.csv(rbind(tab1(), tab2()), file)
  }
)

########################################
output$main_plot <- renderPlot({
  grid.newpage()
  heights <- widths <- c(1.5, rep(1,9), 1.5)/12
  ycenter <- cumsum(c(1.5, rep(1,9), 1.5)/12) - widths/2
  xcenter <- rep(.75/12, 11)
  ypropcenter <- xpropcenter <- numeric(40)

  for (i in 1:11) {
    xpropcenter[i] <- xcenter[i]
    ypropcenter[i] <- ycenter[i]
    vp <- viewport(x = xpropcenter[i], y = ypropcenter[i],
                   height = heights[i], width = 1.5/12)
    pushViewport(vp)
    grid.rect(gp = gpar(fill = propertycolors[i]))
    upViewport()
  }
  ########################################
  vp <- viewport(x = xpropcenter[1], y = ypropcenter[1],
                 height = heights[1], width = 1.5/12)
  pushViewport(vp)
  grid.text("Start", gp = gpar(col = "white", cex = 3))
  upViewport()
  ########################################

  for (i in 2:10) {
    xpropcenter[i + 10] <- ycenter[i]
    ypropcenter[i + 10] <- 1 - xcenter[i]
    vp <- viewport(x = xpropcenter[i + 10], y = ypropcenter[i + 10],
                   height = 1.5/12, width = widths[i])
    pushViewport(vp)
    grid.rect(gp = gpar(fill = propertycolors[10 + i]))
    upViewport()
```

```
  }

  for (i in 1:11) {
    xpropcenter[i + 20] <- 1 - xcenter[i]
    ypropcenter[i + 20] <- rev(ycenter)[i]
    vp <- viewport(x = xpropcenter[i + 20], y = ypropcenter[i + 20],
                   height = heights[i], width = 1.5/12)
    pushViewport(vp)
    grid.rect(gp = gpar(fill = (propertycolors[20 + i])))
    upViewport()
  }

  for (i in 2:10) {
    xpropcenter[i + 30] <- rev(ycenter)[i]
    ypropcenter[i + 30] <- xcenter[i]
    vp <- viewport(x = xpropcenter[i + 30], y = ypropcenter[i + 30],
                   height = 1.5/12, width = widths[i])
    pushViewport(vp)
    grid.rect(gp = gpar(fill = propertycolors[i + 30]))
    upViewport()
  }

  ##############################################
  player <- circleGrob(r = .025, gp = gpar(fill = "white"), name = "player")
  playerland <- editGrob(player, vp = viewport(x = xpropcenter[1],
                                 y = ypropcenter[1]), name = "playerland")
  dicevalue <- textGrob(x = .5, y = .5, "1", name = "dicevalue",
                        gp = gpar(col = 4, cex = 3))

  grid.draw(dicevalue)
  grid.draw(playerland)

  if(is.na(input$nplay1) || input$nplay1 == 0) n1 <- 0 else n1 <- input$nplay1
  if(is.na(input$nplay2) || input$nplay2 == 0) n2 <- 0 else n2 <- input$nplay2
  currentland1 <- land1()[n1 + 1]
  currentland2 <- land2()[n2 + 1]
  currentthrow1 <- land1()[c(n1 + 2, n1 + 3)]
  currentthrow2 <- land2()[c(n2 + 2, n2 + 3)]
  grid.edit("playerland", vp = viewport(x = xpropcenter[currentland1],
                                        y = ypropcenter[currentland1]))
  grid.edit("dicevalue", label = sum(currentthrow1), gp = gpar(col = 4, cex = 3))
  player2land <- editGrob(player, gp = gpar(fill = "red"), vp = viewport(x = xpropcenter[currentland2]
                                                                         y = ypropcenter[currentland2]
  grid.draw(player2land)
})

output$png1 <- downloadHandler(
  filename = 'MonopolyGameBoard.png',
  content = function(file) {
    png(file)
    grid.newpage()
    heights <- widths <- c(1.5, rep(1,9), 1.5)/12
    ycenter <- cumsum(c(1.5, rep(1,9), 1.5)/12) - widths/2
    xcenter <- rep(.75/12, 11)
```

```
ypropcenter <- xpropcenter <- numeric(40)

for (i in 1:11) {
  xpropcenter[i] <- xcenter[i]
  ypropcenter[i] <- ycenter[i]
  vp <- viewport(x = xpropcenter[i], y = ypropcenter[i],
                 height = heights[i], width = 1.5/12)
  pushViewport(vp)
  grid.rect(gp = gpar(fill = propertycolors[i]))
  upViewport()
}
########################################
vp <- viewport(x = xpropcenter[1], y = ypropcenter[1],
               height = heights[1], width = 1.5/12)
pushViewport(vp)
grid.text("Start", gp = gpar(col = "white", cex = 3))
upViewport()
########################################

for (i in 2:10) {
  xpropcenter[i + 10] <- ycenter[i]
  ypropcenter[i + 10] <- 1 - xcenter[i]
  vp <- viewport(x = xpropcenter[i + 10], y = ypropcenter[i + 10],
                 height = 1.5/12, width = widths[i])
  pushViewport(vp)
  grid.rect(gp = gpar(fill = propertycolors[10 + i]))
  upViewport()
}

for (i in 1:11) {
  xpropcenter[i + 20] <- 1 - xcenter[i]
  ypropcenter[i + 20] <- rev(ycenter)[i]
  vp <- viewport(x = xpropcenter[i + 20], y = ypropcenter[i + 20],
                 height = heights[i], width = 1.5/12)
  pushViewport(vp)
  grid.rect(gp = gpar(fill = (propertycolors[20 + i])))
  upViewport()
}

for (i in 2:10) {
  xpropcenter[i + 30] <- rev(ycenter)[i]
  ypropcenter[i + 30] <- xcenter[i]
  vp <- viewport(x = xpropcenter[i + 30], y = ypropcenter[i + 30],
                 height = 1.5/12, width = widths[i])
  pushViewport(vp)
  grid.rect(gp = gpar(fill = propertycolors[i + 30]))
  upViewport()
}

############################################
player <- circleGrob(r = .025, gp = gpar(fill = "white"), name = "player")
playerland <- editGrob(player, vp = viewport(x = xpropcenter[1],
                                             y = ypropcenter[1]), name = "playerland")
dicevalue <- textGrob(x = .5, y = .5, "1", name = "dicevalue",
```

```
                              gp = gpar(col = 4, cex = 3))

      grid.draw(dicevalue)
      grid.draw(playerland)

      if(is.na(input$nplay1) || input$nplay1 == 0) n1 <- 0 else n1 <- input$nplay1
      if(is.na(input$nplay2) || input$nplay2 == 0) n2 <- 0 else n2 <- input$nplay2
      currentland1 <- land1()[n1 + 1]
      currentland2 <- land2()[n2 + 1]
      currentthrow1 <- land1()[c(n1 + 2, n1 + 3)]
      currentthrow2 <- land2()[c(n2 + 2, n2 + 3)]
      grid.edit("playerland", vp = viewport(x = xpropcenter[currentland1],
                                            y = ypropcenter[currentland1]))
      grid.edit("dicevalue", label = sum(currentthrow1), gp = gpar(col = 4, cex = 3))
      player2land <- editGrob(player, gp = gpar(fill = "red"), vp = viewport(x = xpropcenter[currentlan
                                                                             y = ypropcenter[currentlan
      grid.draw(player2land)
      dev.off()
    }
  )

  revenue_analysis <- reactiveVal()
  observeEvent(input$analyze, {
    set.seed(123) # For reproducibility
    total_runs <- 10000000
    lands1 <- numeric(total_runs + 1)
    lands1[1] <- 1
    for(i in 2:(total_runs + 1)) {
      throw <- sample(1:6, size = 2, replace = TRUE)
      lands1[i] <- (lands1[i - 1] + sum(throw) - 1) %% 40 + 1
      if(lands1[i] == 31) lands1[i] <- 11
    }

    freq <- table(factor(lands1[2:(total_runs + 1)], levels = 0:39)) / total_runs

    rev_a <- (950 * (freq[17] + freq[19]) + 1000 * freq[20]) * total_runs
    rev_b <- (1500 * freq[38] + 2000 * freq[40]) * total_runs

    revenue_analysis(list(option_a = rev_a, option_b = rev_b, better = ifelse(rev_a > rev_b, "Option a"
  })

  output$revenue_output <- renderPrint({
    if(is.null(revenue_analysis())) {
      "Click 'Run Revenue Analysis' to see results."
    } else {
      res <- revenue_analysis()
      cat("Expected Revenue Analysis (based on 10M runs):\n")
      cat(sprintf("Option (a) Revenue: $%.2f\n", res$option_a))
      cat(sprintf("Option (b) Revenue: $%.2f\n", res$option_b))
      cat(sprintf("Better Option: %s\n", res$better))
    }
  })

})
```

```
shinyApp(ui = ui, server = server)
#Based on these approximations,
Option (b) yields a higher expected revenue ($825,000,000 vs. $782,500,000),
making it the better choice.
```