Proiect PS Chaos Game

Oprea Tudor, Oprea Mihai Stefan, Mihai Dragos Vasile

2023-02-02

```
library(shiny)
library(shinyBS)
```

Zona de inputuri

```
ui <- shinyUI(
  fluidPage(
    sidebarLayout(
      sidebarPanel(
        sliderInput(
          inputId = "nr_varfuri",
          label = h5("Numar varfuri"),
          min = 3,
          max = 20,
          value = 3,
          step = 1,
          animate = animationOptions(interval = 4000)
        ),
        sliderInput(
          inputId = "numar_noduri",
          label = h5("Numar noduri"),
          min = 1000,
          max = 100000,
          value = 20000,
          step = 100,
          animate = animationOptions(interval = 100)
        ),
        selectizeInput(
          inputId = 'nod_color',
          label = h5("Culorare noduri"),
          choices = list("Colors" = c(
            `Red` = 'red',
            `Blue` = 'blue',
            `DeepPink` = 'deeppink',
            `DarkOrange` = 'darkorange',
            `DarkGreen` = 'darkgreen',
            `Gold` = 'gold',
            `Navy` = 'navy'
          )),
        ),
        selectizeInput(
          inputId = 'shape',
          label = h5("Tip figura geometrica"),
          choices = list("Colors" = c(
            `Polygon` = 'polygon',
            `PolygonModified` = 'polygon_modified',
            `Star` = 'star'
          )),
        ),
        div(bsButton(inputId = "reset", label = "Reset"))
      ),
      mainPanel(div(plotOutput("result")))
    )
  )
)
```

Functia de generare Chaos Game

```
polygon.generate <- function(nr_vertexes){
  dist_noduri = nr_vertexes / (nr_vertexes + 3)  #formula prin care aflam in fct de nr de noduri
ce
  len <- 100000                                  #distanta punem intre ele

coord_endpoint <- matrix(NA, ncol = 2, nrow = nr_vertexes) #creem o matrice pentru endpoints

angle <- 360/nr_vertexes
radius <- 1
```

gasim cate un set de coordonate pentru fiecare endpoint

```
for (i in 1:nr_vertexes) {
  x <- radius * cos(angle * (i-1) * pi/180 + pi/2)
  y <- radius * sin(angle * (i-1) * pi/180 + pi/2)
  coord_endpoint[i, ] <- c(x, y)
}
```

dam fiecarui nod o “categorie”

```
vertexes_ep <- sample(c(1:nr_vertexes), size = len + 1, replace = TRUE)
```

creem matricea de coordonate

```
coord_vertexes <- matrix(NA, ncol = 2, nrow = (len + 1))
coord_vertexes[1, ] <- c(runif(1), runif(1)) #valoarea primului punct
```

calculam coord urmatorului nod in functie de ce categorie are

```
for(i in 1:len){

  endpoint <- vertexes_ep[i]

  x <- coord_endpoint[endpoint, 1]
  y <- coord_endpoint[endpoint, 2]

  x.new <- dist_noduri * x + (1 - dist_noduri) * coord_vertexes[i, 1]
  y.new <- dist_noduri * y + (1 - dist_noduri) * coord_vertexes[i, 2]

  coord_vertexes[i + 1, ] <- c(x.new, y.new)
}

return (list(coord_endpoint, vertexes_ep, coord_vertexes))
```

}

Functia modificata, care genereaza alte figuri geometrice

```
polygon_modified.generate <- function(nr_vertexes){
  dist_noduri = (nr_vertexes - 1) / (nr_vertexes + 2)

len <- 100000

coord_endpoint <- matrix(NA, ncol = 2, nrow = nr_vertexes)

angle <- 360/nr_vertexes
radius <- 1
for (i in 1:nr_vertexes) {
  x <- radius * cos(angle * (i-1) * pi/180 + pi/2)
  y <- radius * sin(angle * (i-1) * pi/180 + pi/2)
  coord_endpoint[i, ] <- c(x, y)
}

vertexes_ep <- sample(c(1:nr_vertexes), size = len + 1, replace = TRUE)

coord_vertexes <- matrix(NA, ncol = 2, nrow = (len + 1))

coord_vertexes[1, ] <- c(runif(1), runif(1))
```

#variabila in care tinem minte categoria predecesorului

```
previous_vertex <- 0
```

cautam ca nodul curent sa nu aiba aceeasi categorie ca predecesorul sau

```
for(i in 1:len){


  while(vertexes_ep[i] == previous_vertex){
    vertexes_ep[i] <- sample(1:nr_vertexes, 1)
  }

  endpoint <- vertexes_ep[i]
  previous_vertex <- vertexes_ep[i]

  x <- coord_endpoint[endpoint, 1]
  y <- coord_endpoint[endpoint, 2]

  x.new <- dist_noduri * x + (1 - dist_noduri) * coord_vertexes[i, 1]
  y.new <- dist_noduri * y + (1 - dist_noduri) * coord_vertexes[i, 2]

  coord_vertexes[i + 1, ] <- c(x.new, y.new)
}

return (list(coord_endpoint, vertexes_ep, coord_vertexes))
```

}

Functia generare stea

vedem la ce distanta sa fie punctele in functie de paritatea numarului de puncte

```
star.generate <- function(nr_vertexes){

if(nr_vertexes %% 2 == 1)
  dist_noduri = nr_vertexes / (nr_vertexes + 5)
else
  dist_noduri = (nr_vertexes - 1) / (nr_vertexes + 4)
len <- 100000
coord_endpoint <- matrix(NA, ncol = 2, nrow = nr_vertexes)

angle <- 360/nr_vertexes
radius <- 1
for (i in 1:nr_vertexes) {
  x <- radius * cos(angle * (i-1) * pi/180 + pi/2)
  y <- radius * sin(angle * (i-1) * pi/180 + pi/2)
  coord_endpoint[i, ] <- c(x, y)
}

vertexes_ep <- sample(c(1:nr_vertexes), size = len + 1, replace = TRUE)

coord_vertexes <- matrix(NA, ncol = 2, nrow = (len + 1))

coord_vertexes[1, ] <- c(runif(1), runif(1))

previous_vertex <- 0

for(i in 1:len){
  #gasim vecinii predecesorului
  previous_vertex_vecin1 <- previous_vertex - 1
  if(previous_vertex_vecin1 <= 0){
    previous_vertex_vecin1 <- nr_vertexes
  }
  previous_vertex_vecin2 <- previous_vertex + 1
  if(previous_vertex_vecin2 > nr_vertexes){
    previous_vertex_vecin2 <- 1
  }
```

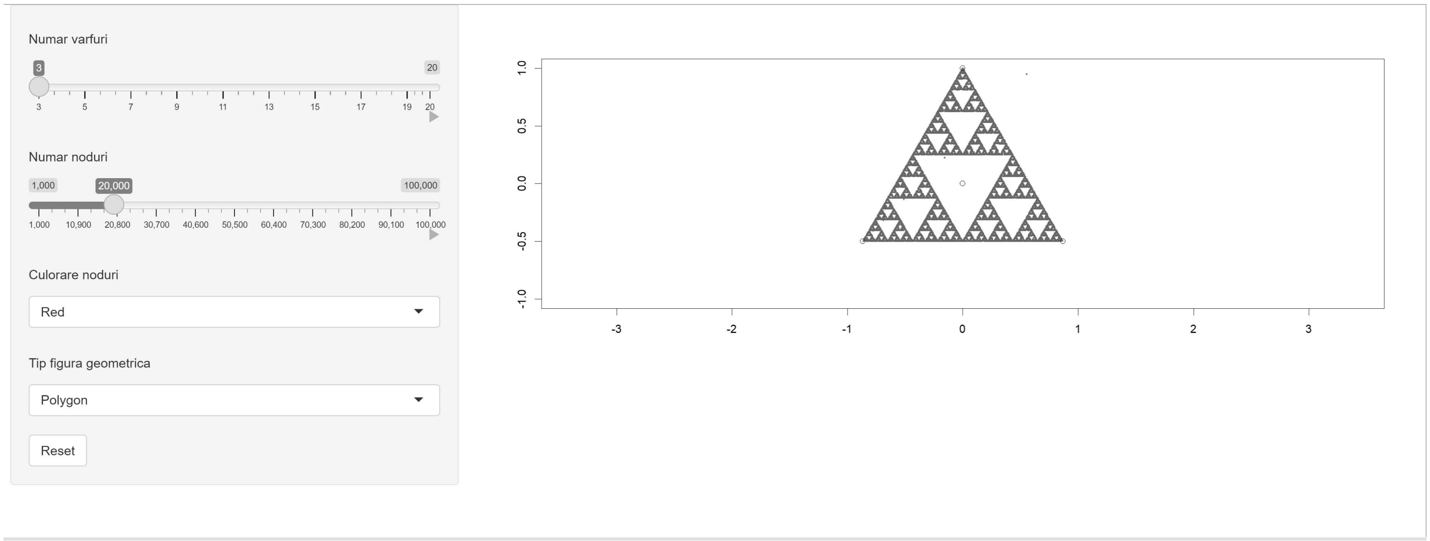
cautam ca nodul curent sa nu aiba aceeasi categorie ca predecesorul sau ori ca vecinii predecesorului

```
  while(vertexes_ep[i] == previous_vertex_vecin2 || vertexes_ep[i] == previous_vertex_vecin1){
    vertexes_ep[i] <- sample(1:nr_vertexes, 1)
  }

  endpoint <- vertexes_ep[i]
  previous_vertex <- vertexes_ep[i]

  x <- coord_endpoint[endpoint, 1]
  y <- coord_endpoint[endpoint, 2]

  x.new <- dist_noduri * x + (1 - dist_noduri) * coord_vertexes[i, 1]
  y.new <- dist_noduri * y + (1 - dist_noduri) * coord_vertexes[i, 2]

  coord_vertexes[i + 1, ] <- c(x.new, y.new)
}

return (list(coord_endpoint, vertexes_ep, coord_vertexes))
```

}

#in functie de optiune, facem forma ceruta

```
server <- shinyServer(function(input, output, session) {
  updateButton(session, inputId = "reset")

all.list <- reactive({
  if(input$reset > -1){
    if(input$shape == 'polygon')
      return(polygon.generate(input$nr_varfuri))
    else if(input$shape == 'polygon_modified')
      return(polygon_modified.generate(input$nr_varfuri))
    else
      return(star.generate(input$nr_varfuri))
  }
})

#afisam pe ecran rezultatul
output$result <- renderPlot({
  coord_endpoint <- all.list()[[1]]
  coord_vertexes <- all.list()[[3]]

  plot(0,0,xlim=c(-1,1),ylim=c(-1,1), asp = 1, xlab = "", ylab = "")

  points(coord_vertexes[1:input$numar_noduri, 1], coord_vertexes[1:input$numar_noduri, 2], pch = ".", cex = 2, col = input$nod_color)

  points(coord_endpoint[, 1], coord_endpoint[, 2], col = "black")
})
})

shinyApp(ui = ui, server = server)
```

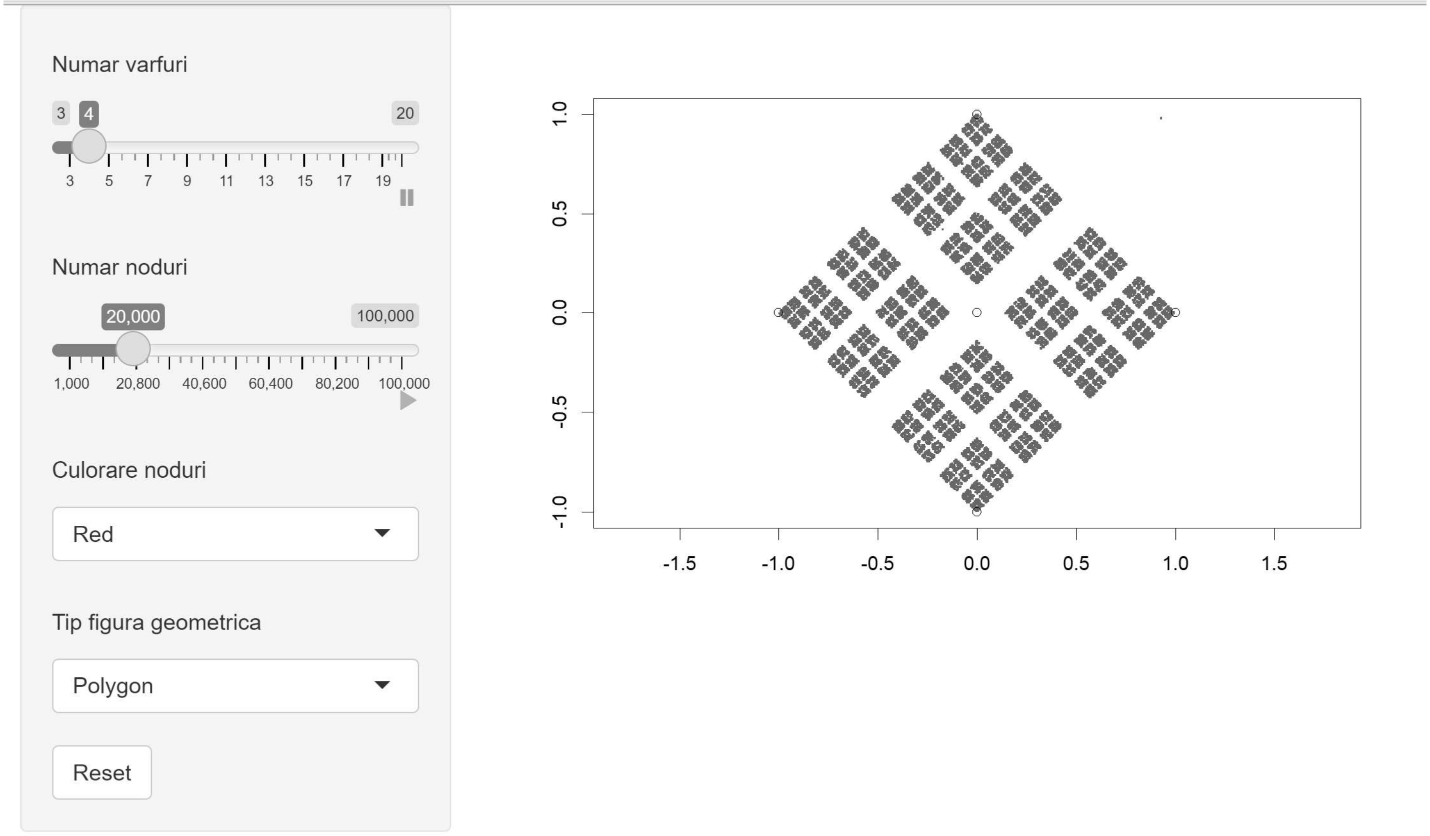


Numar varfuri: 3

Numar noduri: 20,000

Culorare noduri: Red

Tip figura geometrica: Polygon

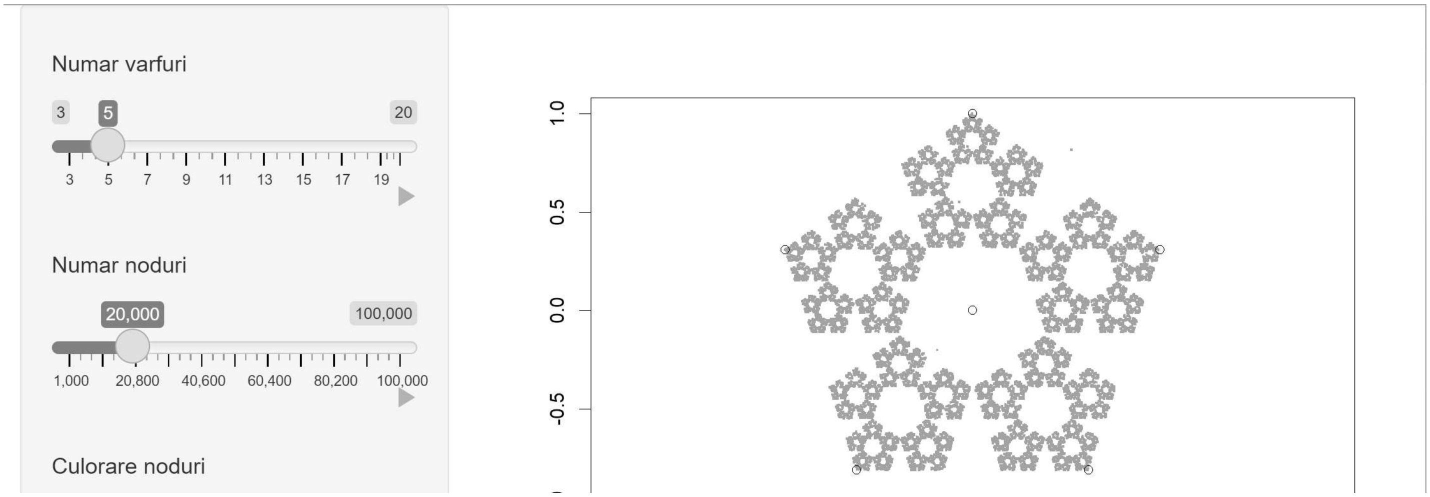


Numar varfuri: 4

Numar noduri: 20,000

Culorare noduri: Red

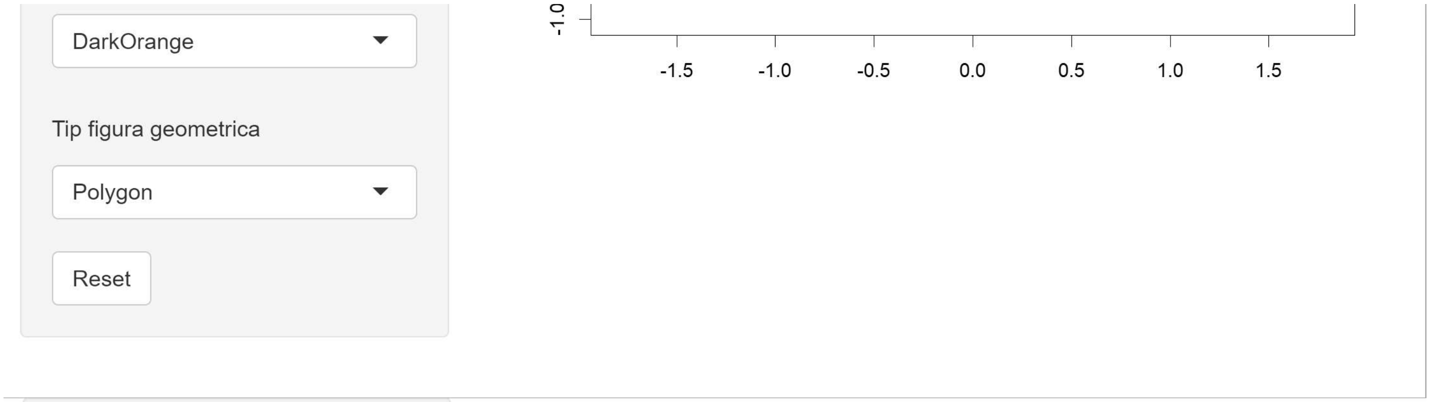
Tip figura geometrica: Polygon



Numar varfuri: 5

Numar noduri: 20,000

Culorare noduri

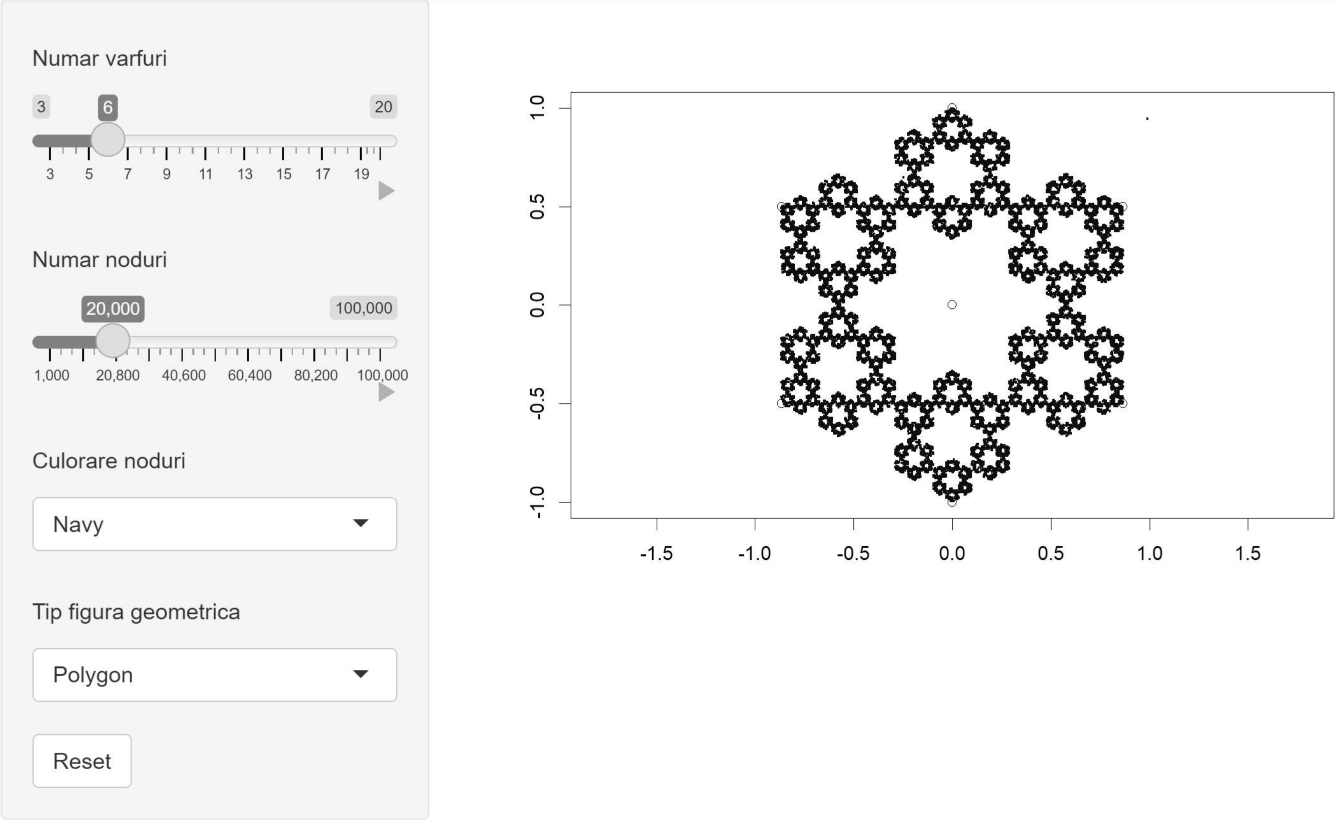


DarkOrange

Tip figura geometrica

Polygon

Reset



Numar varfuri: 6

Numar noduri: 20,000

Culorare noduri

Navy

Tip figura geometrica

Polygon

Reset