# jan2021

2024-10-27

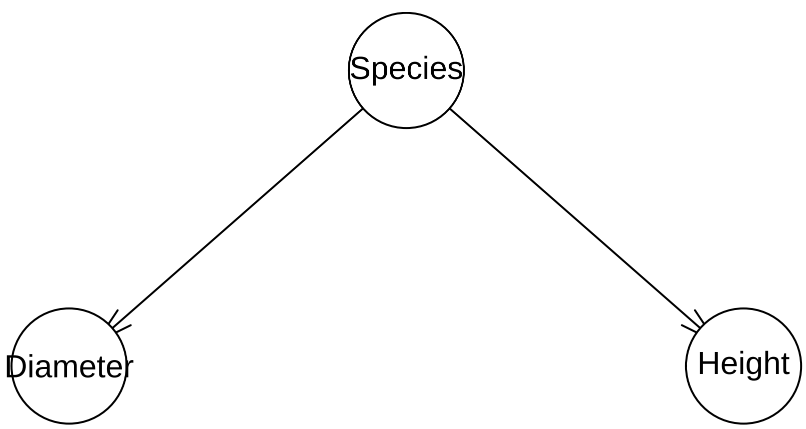
## 1. Graphical Models (6 p) - Using the IC Algorithn

### Overview of the IC Algorithm

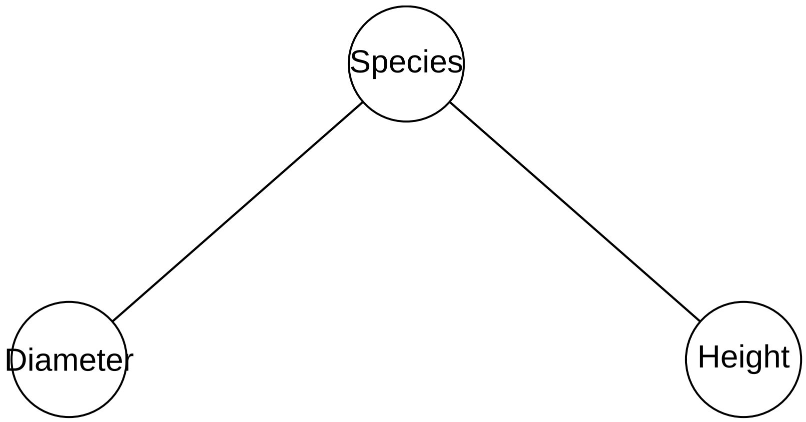
**The IC algorithm consists of three main steps:**

1. *Skeleton Discovery*: Construct an undirected graph that represents dependencies between variables.
2. *Edge Orientation with V-Structures*: Identify causal directions by finding V-structures (patterns where A–> B <– C with no edge between A and C).
3. *Propagation of Orientation*: Further orient edges using rules to avoid cycles and preserve conditional independencies.

```r
library(bnlearn)

data("lizards")

lizardsnet<-model2network("[Species][Diameter|Species][Height|Species]") # True DAG
plot(lizardsnet)
```




```r
plot(cpdag(lizardsnet)) # Plot the true pattern
```




```r
# Independence if p-value is > 0.05

# Skeleton discovery
ci.test(x = "Diameter", y = "Species", data = lizards)  # Keep edge D-S. (Not independent given empty s
```

```
##
##  Mutual Information (disc.)
##
## data:  Diameter ~ Species
## mi = 12.606, df = 1, p-value = 0.0003845
## alternative hypothesis: true value is greater than 0
```

```r
ci.test(x = "Diameter", y = "Height", data = lizards)   # Remove edge D-H. (Independent given empty set
```

```
##
##  Mutual Information (disc.)
##
## data:  Diameter ~ Height
## mi = 0.60771, df = 1, p-value = 0.4357
## alternative hypothesis: true value is greater than 0
```

```r
ci.test(x = "Height", y = "Species", data = lizards)    # Keep edge H-S. (Not independent given empty s
```

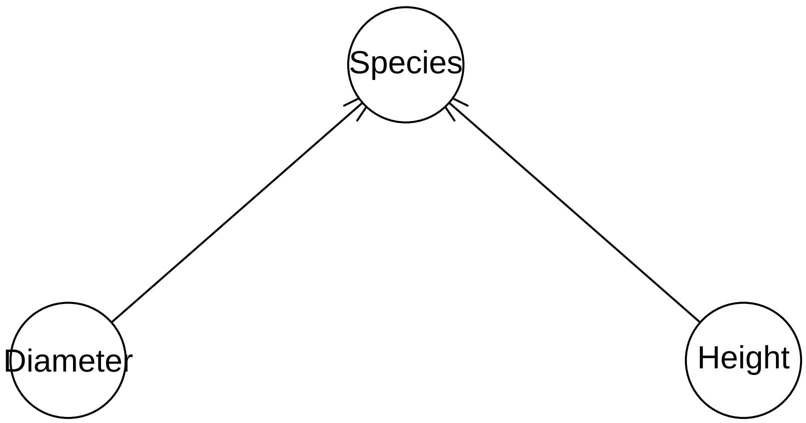
```
##
##  Mutual Information (disc.)
##
## data:  Height ~ Species
## mi = 10.405, df = 1, p-value = 0.001257
## alternative hypothesis: true value is greater than 0
```

```r
# The skeleton now looks like this:
currmod = model2network("[Species][Diameter|Species][Height|Species]")
plot(cpdag(currmod))

# Edge Orientation with V-Structures
# Investigate non adjacent vairables
ci.test(x = "Diameter", y = "Height", z = "Species", data = lizards) # D and H are independent given S
```

```
##
##  Mutual Information (disc.)
##
## data:  Diameter ~ Height | Species
## mi = 2.0256, df = 2, p-value = 0.3632
```

```
## alternative hypothesis: true value is greater than 0
```

```
# Since this test showed that D and H are conditionally independent,
# we choose S as an unsheilded collider: D --> S <-- H


plot(model2network("[Diameter][Height][Species|Diameter:Height]"))
```




# 2. Hidden Markov Models (7 p)

```
library(bnlearn)
library(gRain)
```

```
## Loading required package: gRbase

##
## Attaching package: 'gRbase'

## The following objects are masked from 'package:bnlearn':
##
##     ancestors, children, nodes, parents
```

```
hmm <- model2network("[z0][x0|z0][z1|z0][x1|z1][z2|z1][x2|z2][z3|z2][x3|z3]") # True DAG

states = c("1","2","3","4","5","6","7","8","9","10")
symbols = c("1","2","3","4","5","6","7","8","9","10")

transitionProb = matrix(0,nrow = 10, ncol = 10)
for (j in 1:10) {
  transitionProb[j,j] = 0.5
  transitionProb[j,j%%10 +1 ] = 0.5
}

emissionProb = matrix(0,nrow = 10, ncol = 10)
for (j in 1:10) {
  for (i in 1:5) {
    emissionProb[(j+i-4)%%10+1,j] = 0.2
  }
}

###### Hidden states ######
```

```r
cptZ0 = rep(0.1,10)
dim(cptZ0) = c(10)
dimnames(cptZ0) = list(states)

cptZ1 = transitionProb
dim(cptZ1) = c(10,10)
dimnames(cptZ1) = list("z1" = states, "z0" = states)


cptZ2 = transitionProb
dim(cptZ2) = c(10,10)
dimnames(cptZ2) = list("z2" = states, "z1" = states)

cptZ3 = transitionProb
dim(cptZ3) = c(10,10)
dimnames(cptZ3) = list("z3" = states, "z2" = states)
#############################

###### observed states ######
cptX0 = emissionProb
dim(cptX0) = c(10,10)
dimnames(cptX0) = list("x0" = symbols, "z0" = states)

cptX1 = emissionProb
dim(cptX1) = c(10,10)
dimnames(cptX1) = list("x1" = symbols, "z1" = states)

cptX2 = emissionProb
dim(cptX2) = c(10,10)
dimnames(cptX2) = list("x2" = symbols, "z2" = states)

cptX3 = emissionProb
dim(cptX3) = c(10,10)
dimnames(cptX3) = list("x3" = symbols, "z3" = states)
#############################

nodes = c("z0", "z1", "z2", "z3", "x0", "x1", "x2", "x3")
hmmfit = custom.fit(hmm, list(z0=cptZ0, z1=cptZ1, z2=cptZ2, z3=cptZ3, x0=cptX0, x1=cptX1, x2=cptX2, x3=
compiledgrain = compile(as.grain(hmmfit))


querygrain(setEvidence(compiledgrain,nodes = c("x0", "x2"), states = c("1","3")), nodes = "z0")$z0
```

```
## z0
##     1     2     3     4     5     6     7     8     9    10
## 0.125 0.375 0.500 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```r
querygrain(setEvidence(compiledgrain,nodes = c("x0", "x2"), states = c("1","3")), nodes = "z1")$z1
```

```
## z1
##    1    2    3    4    5    6    7    8    9   10
## 0.25 0.50 0.25 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

```r
querygrain(setEvidence(compiledgrain,nodes = c("x0", "x2"), states = c("1","3")), nodes = "z2")$z2
```

```
## z2
##     1     2     3     4     5     6     7     8     9    10
## 0.500 0.375 0.125 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```
querygrain(setEvidence(compiledgrain,nodes = c("x0", "x2"), states = c("1","3")), nodes = "z3")$z3
```

```
## z3
##      1      2      3      4      5      6      7      8      9     10
## 0.4375 0.2500 0.0625 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.2500
```

# 3. Reinforcement Learning (7 p)

```
set.seed(1234)
library(ggplot2)

arrows <- c("^", ">", "v", "<")
action_deltas <- list(c(1,0), # up
                      c(0,1), # right
                      c(-1,0), # down
                      c(0,-1)) # left

vis_environment <- function(iterations=0, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){


  df <- expand.grid(x=1:H,y=1:W)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,1],NA),df$x,df$y)
  df$val1 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,2],NA),df$x,df$y)
  df$val2 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,3],NA),df$x,df$y)
  df$val3 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,4],NA),df$x,df$y)
  df$val4 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y)
    ifelse(reward_map[x,y] == 0,arrows[GreedyPolicy(x,y)],reward_map[x,y]),df$x,df$y)
  df$val5 <- as.vector(foo)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,max(q_table[x,y,]),
                                     ifelse(reward_map[x,y]<0,NA,reward_map[x,y])),df$x,df$y)
  df$val6 <- as.vector(foo)

  print(ggplot(df,aes(x = y,y = x)) +
          scale_fill_gradient(low = "white", high = "green", na.value = "red", name = "") +
          geom_tile(aes(fill=val6)) +
          geom_text(aes(label = val1),size = 4,nudge_y = .35,na.rm = TRUE) +
          geom_text(aes(label = val2),size = 4,nudge_x = .35,na.rm = TRUE) +
          geom_text(aes(label = val3),size = 4,nudge_y = -.35,na.rm = TRUE) +
          geom_text(aes(label = val4),size = 4,nudge_x = -.35,na.rm = TRUE) +
          geom_text(aes(label = val5),size = 10) +
          geom_tile(fill = 'transparent', colour = 'black') +
          ggtitle(paste("Q-table after ",iterations," iterations\n",
                        "(epsilon = ",epsilon,", alpha = ",alpha,"gamma = ",
                        gamma,", beta = ",beta,")")) +
          theme(plot.title = element_text(hjust = 0.5)) +
          scale_x_continuous(breaks = c(1:W),labels = c(1:W)) +
```

```r
          scale_y_continuous(breaks = c(1:H),labels = c(1:H)))

}

GreedyPolicy <- function(x, y){

  q_values = q_table[x, y, ]

  # Find all actions with the maximum Q-value
  max_actions = which(q_values == max(q_values))
  if (length(max_actions) == 1) {
    return(max_actions)
  } else {
    return(sample(max_actions, 1))
  }

}

EpsilonGreedyPolicy <- function(x, y, epsilon){

  # Your code here.
  if (runif(1) < epsilon) {
    return (sample(1:4,1))
  } else {
    return (GreedyPolicy(x,y))
  }
}

transition_model <- function(x, y, action, beta){

  delta <- sample(-1:1, size = 1, prob = c(0.5*beta,1-beta,0.5*beta))
  final_action <- ((action + delta + 3) %% 4) + 1
  foo <- c(x,y) + unlist(action_deltas[final_action])
  foo <- pmax(c(1,1),pmin(foo,c(H,W)))

  return (foo)
}

q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0, tr = 1){
  Q = start_state
  x = Q[1]
  y = Q[2]
  episode_correction = 0
  ite = 0
  repeat{

    # Follow policy, execute action, get reward.
    action = EpsilonGreedyPolicy(x,y,epsilon*tr) # follow policy
    next_state = transition_model(x,y,action,beta) # excecute action
    reward = reward_map[next_state[1],next_state[2]] # get reward

    # Q-table update.
```

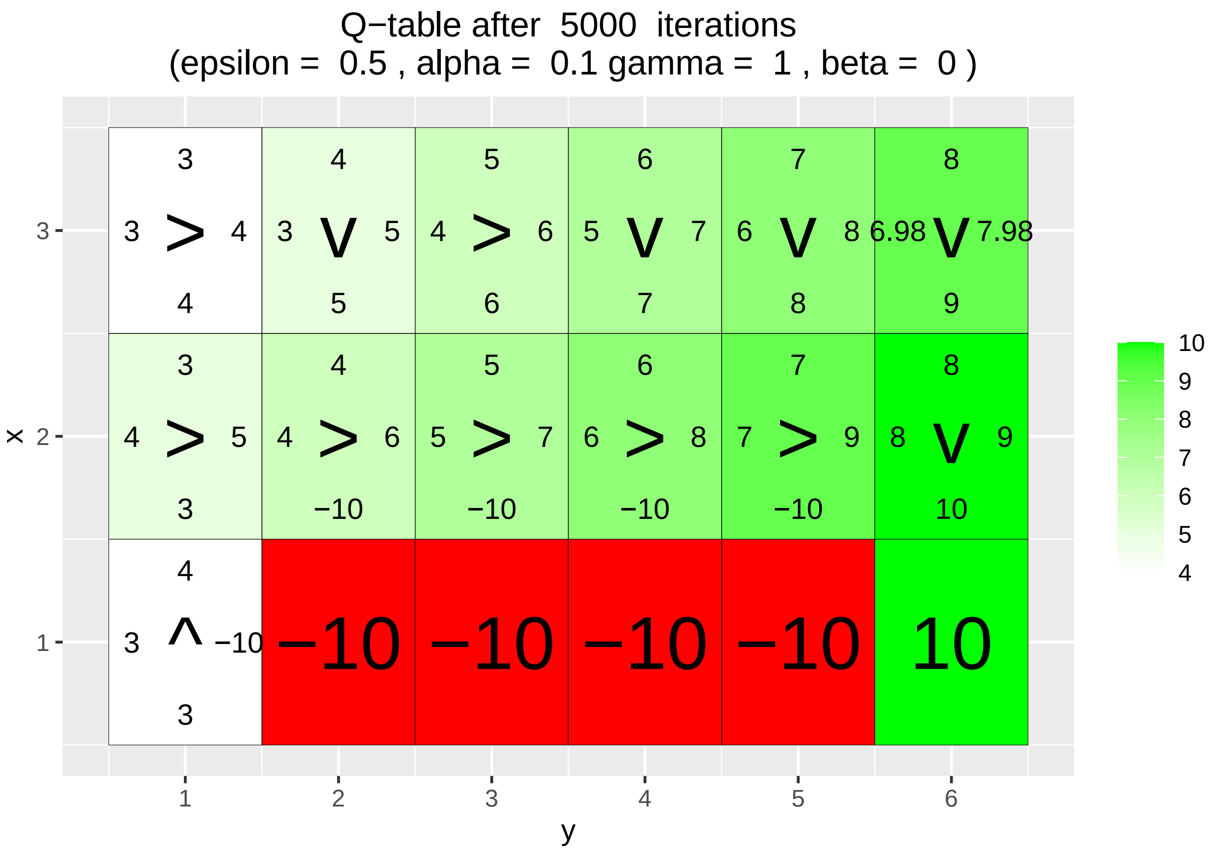
```r
    correction = ifelse(reward == 0, -1, reward) + gamma * max(q_table[next_state[1],next_state[2],])-q
    q_table[x,y,action] <<- q_table[x,y,action] + alpha * (correction*tr)
    episode_correction = episode_correction + correction*tr

    x = next_state[1]
    y = next_state[2]

    if(reward!=0)
      # End episode.
      return (c(reward - ite,episode_correction))
    else
      ite = ite + 1
  }

}

SARSA <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){
  Q = start_state
  x = Q[1]
  y = Q[2]
  episode_correction = 0
  ite = 0
  #next_action = EpsilonGreedyPolicy(x,y,epsilon) # follow policy
  repeat{

    # S A R S A
    # Follow policy, execute action, get reward.
    action = EpsilonGreedyPolicy(x,y,epsilon)
    next_state = transition_model(x,y,action,beta) # excecute action
    reward = reward_map[next_state[1],next_state[2]] # get reward
    next_action = EpsilonGreedyPolicy(next_state[1],next_state[2],epsilon) # follow policy

    # Q-table update.
    correction = ifelse(reward==0,-1,reward) + gamma * (q_table[next_state[1],next_state[2],next_action]
    q_table[x,y,action] <<- q_table[x,y,action] + alpha * (correction)
    episode_correction = episode_correction + correction

    x = next_state[1]
    y = next_state[2]

    if(reward!=0) {
      # End episode.
      return (c(reward - ite, episode_correction))
    }
    else {
      ite = ite + 1
    }
  }

}


# SARSASA <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){
```

```
#   s = start_state
#   episode_correction = 0
#   ite = 0
#   repeat{
#     # S A R S A
#     # Follow policy, execute action, get reward.
#     a = EpsilonGreedyPolicy(s[1],s[2],epsilon) # find best action a
#     sP = transition_model(s[1],s[2],a,beta) # get next state s'
#     r = reward_map[sP[1],sP[2]] # get reward r
#     aP = EpsilonGreedyPolicy(sP[1],sP[2],epsilon)   # find action a'
#
#     if (r!=0) { # Break
#       correction = ifelse(r==0,-1,r) - q_table[s[1],s[2],a]
#       q_table[s[1],s[2],a] <<- q_table[s[1],s[2],a] + alpha * (correction)
#       break
#     } else { # Do next action, reward
#       sPP = transition_model(sP[1],sP[2],aP,beta)     # get state s''
#       rP = reward_map[sPP[1],sPP[2]]                  # get reward r'
#       if (rP!=0) { # Break
#         correction = ifelse(r==0,-1,r) + gamma * (q_table[sP[1],sP[2],aP]) - q_table[s[1], s[2], a]
#         q_table[s[1], s[2], a] <<- q_table[s[1], s[2], a] + alpha * correction
#       } else { # Do next action, reward
#         aPP = EpsilonGreedyPolicy(sPP[1],sPP[2],epsilon)  # find action a''
#         # Update Q table
#         correction = ifelse(r==0,-1,r) + gamma*ifelse(rP==0,-1,rP) + (gamma**2) * (q_table[sPP[1],sPP
#         q_table[s[1],s[2],a] <<- q_table[s[1],s[2],a] + alpha * (correction)
#       }
#     }
#
#     s[1] = sP[1]
#     s[2] = sP[2]
#
#     if(r!=0) {
#       # End episode.
#       return (c(r - ite, 0))
#     }
#     if(rP!=0)
#       return (c(rP-ite,0))
#     else {
#       ite = ite + 1
#     }
#   }
#
# }

SARSASA <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){
  # Initialize first state-action pair
  s = start_state
  a = EpsilonGreedyPolicy(s[1], s[2], epsilon)
  ite = 0

  # Get next state, reward, and action
  sP = transition_model(s[1], s[2], a, beta)
```

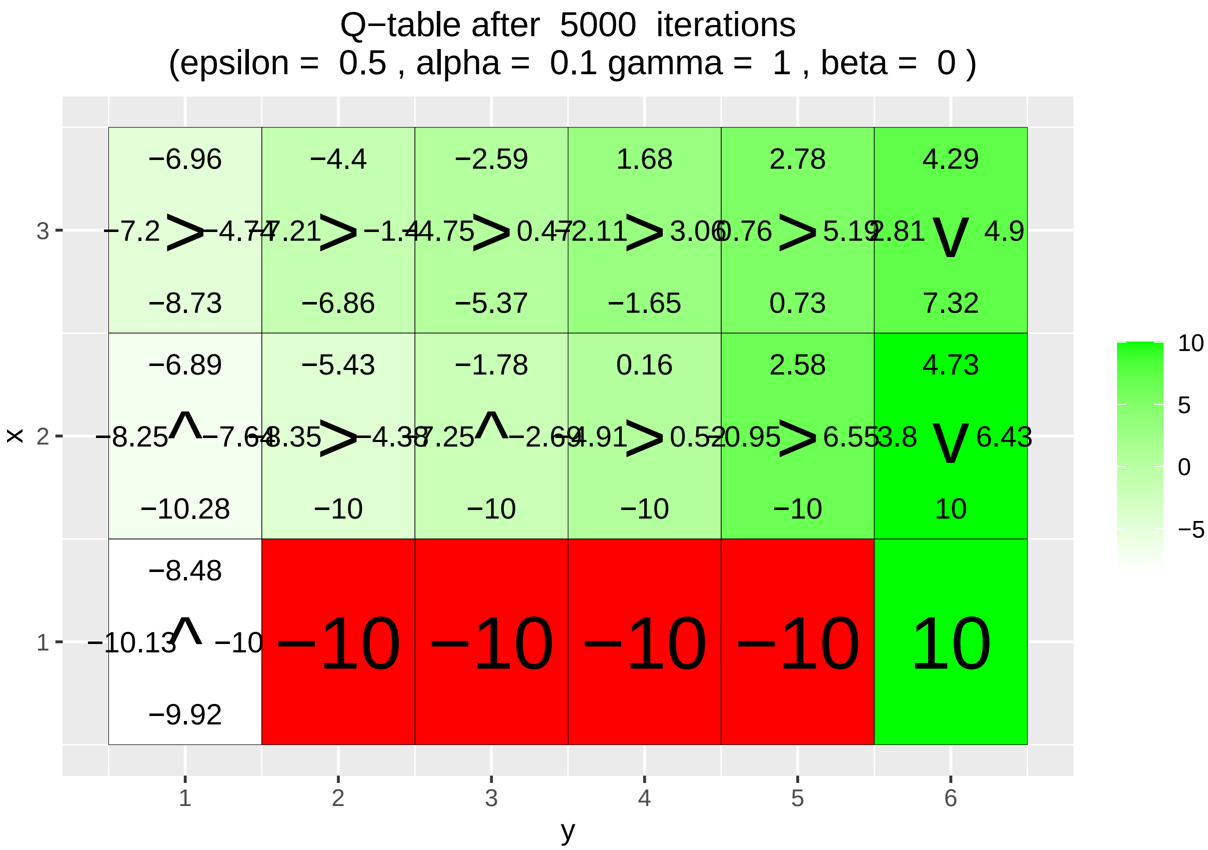
```
  r = reward_map[sP[1], sP[2]]
  aP = EpsilonGreedyPolicy(sP[1], sP[2], epsilon)

  repeat{
    if(r != 0){
      # Terminal state - update current state-action only
      old_q = q_table[s[1], s[2], a]
      q_table[s[1], s[2], a] <<- old_q + alpha*(r - old_q)
      return(c(r - ite, 0))
    }
    else{
      # Get next state-action pair and reward
      sPP = transition_model(sP[1], sP[2], aP, beta)
      rP = reward_map[sPP[1], sPP[2]]
      aPP = EpsilonGreedyPolicy(sPP[1], sPP[2], epsilon)

      if(rP != 0){
        # Next state is terminal - update both current and next state Q-values
        old_q = q_table[s[1], s[2], a]
        q_table[s[1], s[2], a] <<- old_q + alpha*(ifelse(r==0,-1,r) + gamma*rP - old_q)

        old_q = q_table[sP[1], sP[2], aP]
        q_table[sP[1], sP[2], aP] <<- old_q + alpha*(rP - old_q)
        return(c(rP - ite - 1, 0))
      }
      else{
        # Update Q-value and move to next state
        old_q = q_table[s[1], s[2], a]
        q_table[s[1], s[2], a] <<- old_q +
          alpha*(ifelse(r==0,-1,r) + gamma*ifelse(rP==0,-1,rP) + gamma*gamma*q_table[sPP[1], sPP[2], aP

        # Move to next state
        s = sP
        a = aP
        sP = sPP
        r = rP
        aP = aPP
        ite = ite + 1
      }
    }
  }
}

SARSA2 <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){ # JOSES code just

  cur_pos <- start_state
  cur_action <- EpsilonGreedyPolicy(cur_pos[1], cur_pos[2], epsilon)

  # Follow policy, execute action, get reward.
  new_pos <- transition_model(cur_pos[1], cur_pos[2], cur_action, beta)
  reward <- reward_map[new_pos[1], new_pos[2]]
  new_action <- EpsilonGreedyPolicy(new_pos[1], new_pos[2], epsilon)
```

```
  repeat{
    if(reward!=0){
      # End episode.
      old_q <- q_table[cur_pos[1], cur_pos[2], cur_action]
      q_table[cur_pos[1], cur_pos[2], cur_action] <<- old_q + alpha*(reward - old_q)
      break
    }
    else{
      # Follow policy, execute action, get reward.
      new_pos2 <- transition_model(new_pos[1], new_pos[2], new_action, beta)
      reward2 <- reward_map[new_pos2[1], new_pos2[2]]
      new_action2 <- EpsilonGreedyPolicy(new_pos2[1], new_pos2[2], epsilon)

      if(reward2!=0){
        # End episode.
        old_q <- q_table[cur_pos[1], cur_pos[2], cur_action]
        q_table[cur_pos[1], cur_pos[2], cur_action] <<- old_q + alpha*(-1+gamma*reward2 - old_q)
        old_q <- q_table[new_pos[1], new_pos[2], new_action]
        q_table[new_pos[1], new_pos[2], new_action] <<- old_q + alpha*(reward2 - old_q)
        break
      }
      else{
        old_q <- q_table[cur_pos[1], cur_pos[2], cur_action]
        q_table[cur_pos[1], cur_pos[2], cur_action] <<- old_q + alpha*(-1+gamma*(-1)+gamma*gamma*q_tabl
        cur_pos <- new_pos
        cur_action <- new_action
        new_pos <- new_pos2
        reward <- reward2
        new_action <- new_action2
      }
    }
  }

}


MovingAverage <- function(x, n){
  cx <- c(0,cumsum(x))
  rsum <- (cx[(n+1):length(cx)] - cx[1:(length(cx) - n)]) / n
  return (rsum)
}


# Environment C (the effect of beta).

H <- 3
W <- 6

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,2:5] <- -10
reward_map[1,6] <- 10

### Q LEARNING ###
```

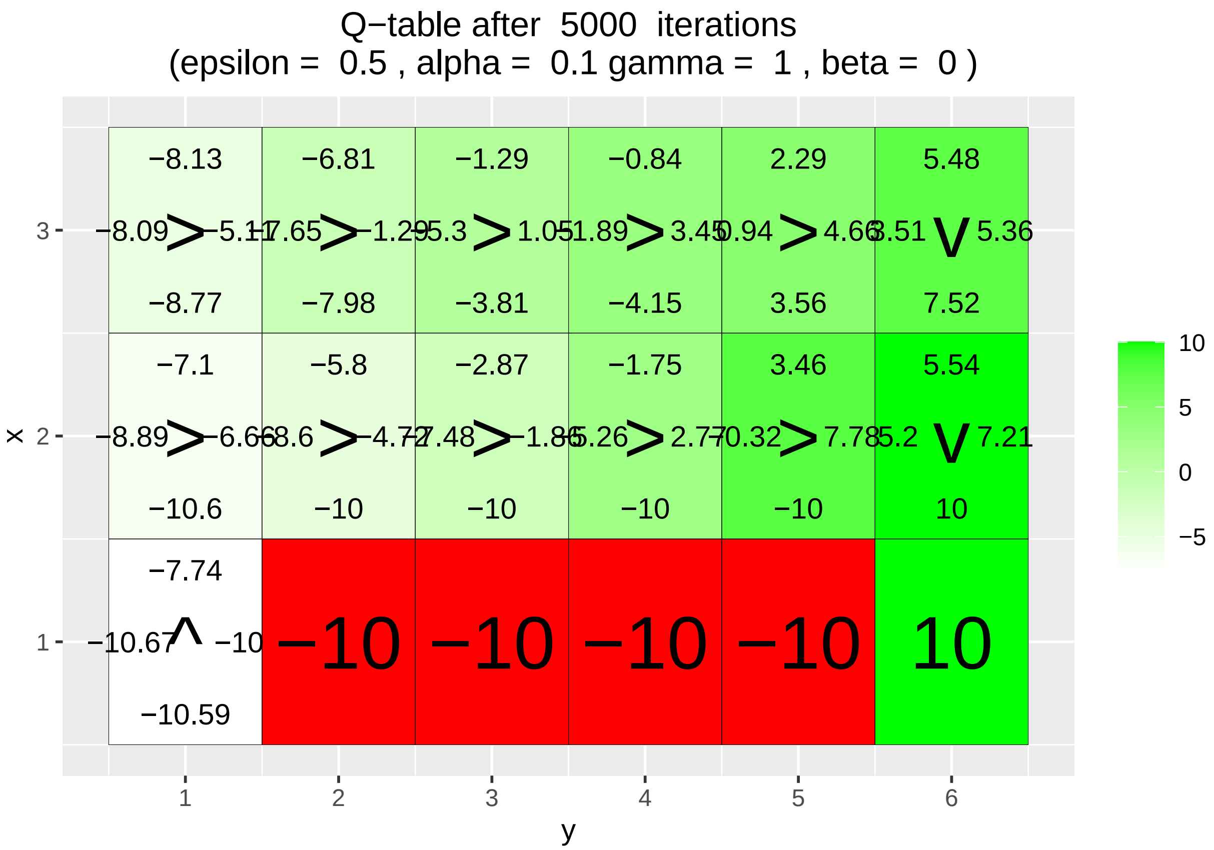
```r
q_table <- array(0,dim = c(H,W,4))
rewardQ = NULL
for(i in 1:5000) {
  foo <- q_learning(epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1, start_state = c(1,1))
  rewardQ = c(rewardQ,foo[1])
}
vis_environment(i, epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1)
```



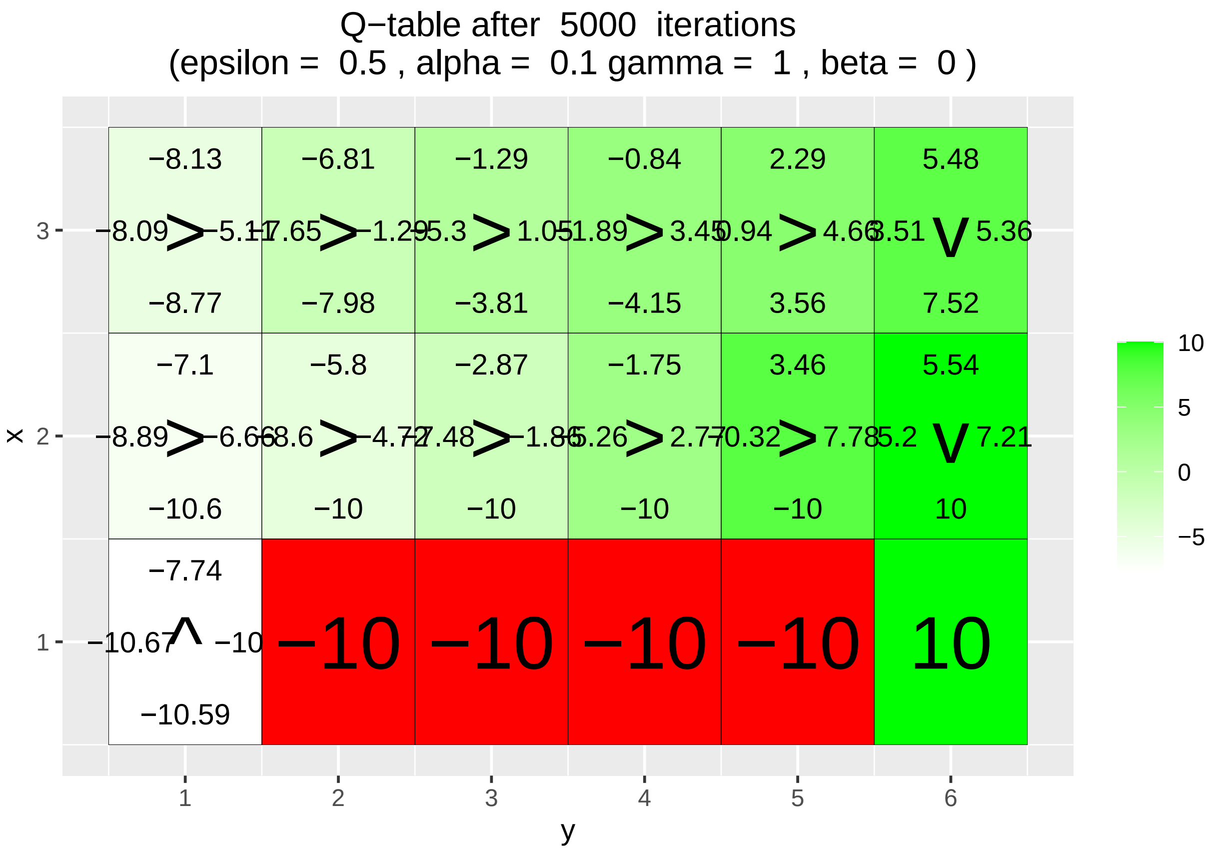

```r
### SARSA ###
q_table <- array(0,dim = c(H,W,4))
rewardS = NULL
for(i in 1:5000) {
  foo <- SARSA(epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1, start_state = c(1,1))
  rewardS = c(rewardS,foo[1])
}
vis_environment(i, epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1)
```




```r
### SARSASA ###
q_table <- array(0,dim = c(H,W,4))
set.seed(123)
rewardSA = NULL
for(i in 1:5000) {
  foo <- SARSASA(epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1, start_state = c(1,1))
  rewardSA = c(rewardSA,foo[1])
}
vis_environment(i, epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1)
```




```r
### SARSASAJ ###
q_table <- array(0,dim = c(H,W,4))
set.seed(123)
for(i in 1:5000) {
  foo <- SARSA2(epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1, start_state = c(1,1))
}
vis_environment(i, epsilon = 0.5, gamma = 1, beta = 0, alpha = 0.1)
```




```
plot(MovingAverage(rewardQ,100),type = "l", col = "green", ylim = c(-15,-5))
lines(MovingAverage(rewardS,100),type = "l", col = "blue")
lines(MovingAverage(rewardSA,100),type = "l", col = "red")
```