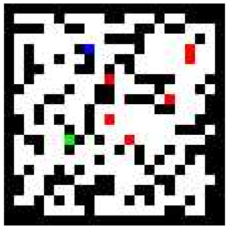
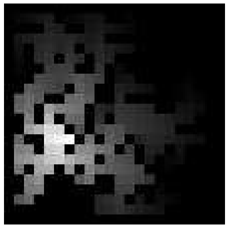
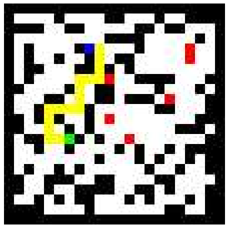
```julia
begin
    using PlutoUI
    using Plots
    using Images
    using LinearAlgebra
end
```



```julia
# Generate Grid_World
begin
    m, n, p1, p2= 20, 20, 200, 6 # row, column, block index (the range of random), number of trap
    Grid_List, Block_List, Trap_List, start, finish, Grid_Map = Generate_Map(m,n,p1,p2)
    img=show_RGB(Grid_List, Trap_List, start, finish, m, n)
end
```



```julia
# Generate Value Map
begin
    Value_List=Value_Iteration(Grid_List,Trap_List,finish,Grid_Map,m,n)
    Value_Map=Gray.([zeros(Int64,1,n+2)
            zeros(Int64,m,1) reshape(Value_List,m,:) zeros(Int64,m,1)
            zeros(Int64,1,n+2)])
end
```



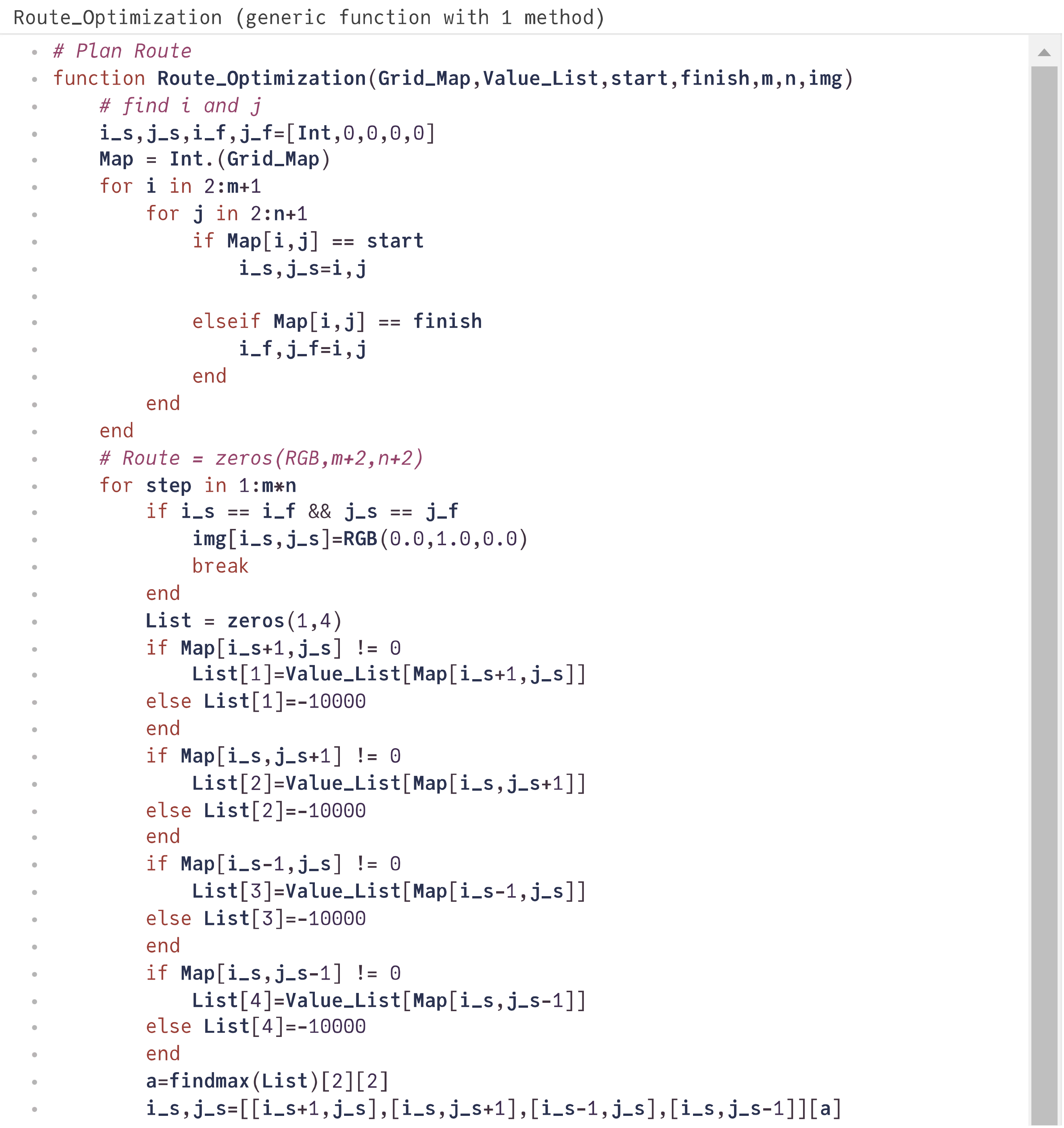
```
# Generate Route
begin
    Route_Optimization(Grid_Map,Value_List,start,finish,m,n,img)
end
```

```
Value_Iteration (generic function with 1 method)
```

```
# Value Iteration
function Value_Iteration(Grid_List,Trap_List,finish,Grid_Map,m,n)
    greedy = 0.85
    discounting = 0.98
    reward = -0.01
    # Initialize Value
    Value_List = 0.0*Grid_List

    for step in 1:100
    Value_List[finish] = 1
    Value_List[Trap_List] .= -0.5
    Value_Map = [zeros(1,n+2)
                zeros(m,1) reshape(Value_List,m,:) zeros(m,1)
                zeros(1,n+2)]

    # Value Iteratoin
    new_Value_List = 0.0*Grid_List
    for i in 2:m+1
        for j in 2:n+1
            if Grid_Map[i,j] != 0
                # Define Value_Vec
                Value_Vec = zeros(1,4)
                if Grid_Map[i+1,j] != 0
                    Value_Vec[1] = Value_Map[i+1,j]
                end
                if Grid_Map[i,j+1] != 0
                    Value_Vec[2] = Value_Map[i,j+1]
                end
                if Grid_Map[i-1,j] != 0
                    Value_Vec[3] = Value_Map[i-1,j]
                end
                if Grid_Map[i,j-1] != 0
                    Value_Vec[4] = Value_Map[i,j-1]
                end
                index = findmax(Value_Vec)[2][2]
                List = [4 1 2 3 4 1]
                Direction_Vec = zeros(1,4)
                Direction_Vec[List[index+1]] = greedy
                Direction_Vec[List[index]] = 0.5*(1-greedy)
                Direction_Vec[List[index+2]] = 0.5*(1-greedy)
                new_Value_List[Grid_Map[i,j]] = discounting*sum(Direction_Vec.*Value_Vec)+reward
            end
        end
    end
        Value_List = new_Value_List
    end
    Value_List[finish] = 1
    Value_List[Trap_List] .= -0.5
    return Value_List
end
```



```
Route_Optimization (generic function with 1 method)
```

```
# Plan Route
function Route_Optimization(Grid_Map,Value_List,start,finish,m,n,img)
    # find i and j
    i_s,j_s,i_f,j_f=[Int,0,0,0,0]
    Map = Int.(Grid_Map)
    for i in 2:m+1
        for j in 2:n+1
            if Map[i,j] == start
                i_s,j_s=i,j

            elseif Map[i,j] == finish
                i_f,j_f=i,j
            end
        end
    end
    # Route = zeros(RGB,m+2,n+2)
    for step in 1:m*n
        if i_s == i_f && j_s == j_f
            img[i_s,j_s]=RGB(0.0,1.0,0.0)
            break
        end
        List = zeros(1,4)
        if Map[i_s+1,j_s] != 0
            List[1]=Value_List[Map[i_s+1,j_s]]
        else List[1]=-10000
        end
        if Map[i_s,j_s+1] != 0
            List[2]=Value_List[Map[i_s,j_s+1]]
        else List[2]=-10000
        end
        if Map[i_s-1,j_s] != 0
            List[3]=Value_List[Map[i_s-1,j_s]]
        else List[3]=-10000
        end
        if Map[i_s,j_s-1] != 0
            List[4]=Value_List[Map[i_s,j_s-1]]
        else List[4]=-10000
        end
        a=findmax(List)[2][2]
        i_s,j_s=[[i_s+1,j_s],[i_s,j_s+1],[i_s-1,j_s],[i_s,j_s-1]][a]
        img[i_s,j_s]=RGB(1.0,1.0,0.0)
    end
    return img
    # move
    # till reach the end
end
```

```
show_RGB (generic function with 1 method)
```

```
# Show RGB Result
function show_RGB(Grid_List, Trap_List, start, finish, m, n)
    Grid_List[Trap_List] .= -1
    Grid_List[start] = -2
    Grid_List[finish] = -3
    Map=[zeros(Int64,1,n+2)
                zeros(Int64,m,1) reshape(Grid_List,m,:) zeros(Int64,m,1)
                zeros(Int64,1,n+2)]
    img = zeros(RGB,m+2,n+2)
    for i in 1:m+2
        for j in 1:n+2
            if Map[i,j] > 0
                img[i,j] = RGB(1.0,1.0,1.0)
            elseif Map[i,j] == 0
                img[i,j] = RGB(0.0,0.0,0.0)
            elseif Map[i,j] == -1
                img[i,j] = RGB(1.0,0.0,0.0)
            elseif Map[i,j] == -2
                img[i,j] = RGB(0.0,0.0,1.0)
            elseif Map[i,j] == -3
                img[i,j] = RGB(0.0,1.0,0.0)
            end
        end
    end
    return img
end
```

```
Generate_Map (generic function with 1 method)
```

```
# Generate Map
function Generate_Map(m,n,p1,p2)
    Grid_List = []
    for i in 1:m*n
        append!(Grid_List,i)
    end
    # Define Boundary
    Block_List = []
    for i in 1:rand(1:p1)
        r = rand(1:m*n)
        append!(Block_List,r)
        Grid_List[r] = 0
    end
    # Generate start, finish
    start = generate_start(Grid_List)
    finish = generate_finish(Grid_List, start)
    # Generate trap
    Trap_List = []
    for i in 1:p2
        r = rand(Grid_List)
        if r != start && r != finish && r != 0
            append!(Trap_List,r)
        end
    end

    Grid_Map = [zeros(Int64,1,n+2)
                zeros(Int64,m,1) reshape(Grid_List,m,:) zeros(Int64,m,1)
                zeros(Int64,1,n+2)]
    return Grid_List,Block_List,Trap_List,start,finish,Grid_Map
end
```

```
generate_start (generic function with 1 method)
```

```julia
# Random choose start from avaialbe Grid
function generate_start(Grid_List)
    start = 0
    while start == 0
        start = rand(Grid_List)
    end
    return start
end
```

```
generate_finish (generic function with 1 method)
```

```julia
# Random choose destination from avaialbe Grid (except start)
function generate_finish(Grid_List, start)
    finish = 0
    while finish == 0 || finish == start
          finish = rand(Grid_List)
    end
    return finish
end
```

```
step (generic function with 1 method)
```

```julia
function step(x)
    if x>0
        return 1.0
    else
        return 0.0
    end
end
```