```
using DMUStudent.HW4: HW4, gw, render
using POMDPModels: SimpleGridWorld
using LinearAlgebra: I
using CommonRLInterface: actions, act!, observe, reset!, AbstractEnv,
observations, terminated, clone
import POMDPTools
using SparseArrays
using Statistics: mean

function sarsa_episode!(Q, env; ϵ=0.10, γ=0.99, α=0.2)
    start = time()

    function policy(s)
        if rand() < ϵ
            return rand(actions(env))
        else
            return argmax(a->Q[(s, a)], actions(env))
        end
    end

    s = observe(env)
    a = policy(s)
    r = act!(env, a)
    sp = observe(env)
    hist = [s]

    while !terminated(env)
        ap = policy(sp)

        Q[(s,a)] += α*(r + γ*Q[(sp, ap)] - Q[(s, a)])

        s = sp
        a = ap
        r = act!(env, a)
        sp = observe(env)
        push!(hist, sp)
    end

    Q[(s,a)] += α*(r - Q[(s, a)])

    return (hist=hist, Q = copy(Q), time=time()-start)
end

function sarsa!(env; n_episodes=100_000)
    Q = Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    episodes = []

    for i in 1:n_episodes
        reset!(env)
```

```
        push!(episodes, sarsa_episode!(Q, env;
                                       ϵ=max(0.1, 1-i/n_episodes)))
    end

    return episodes
end

function sarsa_lambda_episode!(Q, env; ϵ=0.10, γ=0.99, α=0.05, λ=0.9)

    start = time()

    function policy(s)
        if rand() < ϵ
            return rand(actions(env))
        else
            return argmax(a->Q[(s, a)], actions(env))
        end
    end

    s = observe(env)
    a = policy(s)
    r = act!(env, a)
    sp = observe(env)
    hist = [s]
    N = Dict((s, a) => 0.0)

    while !terminated(env)
        ap = policy(sp)

        N[(s, a)] = get(N, (s, a), 0.0) + 1

        δ = r + γ*Q[(sp, ap)] - Q[(s, a)]

        for ((s, a), n) in N
            Q[(s, a)] += α*δ*n
            N[(s, a)] *= γ*λ
        end

        s = sp
        a = ap
        r = act!(env, a)
        sp = observe(env)
        push!(hist, sp)
    end

    N[(s, a)] = get(N, (s, a), 0.0) + 1
    δ = r - Q[(s, a)]

    for ((s, a), n) in N
        Q[(s, a)] += α*δ*n
```

```
        N[(s, a)] *= γ*λ
    end

    return (hist=hist, Q = copy(Q), time=time()-start)
end

function sarsa_lambda!(env; n_episodes=100_000, kwargs...)
    Q = Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    episodes = []

    for i in 1:n_episodes
        reset!(env)
        push!(episodes, sarsa_lambda_episode!(Q, env;
                                              ϵ=max(0.01, 1-i/
n_episodes),
                                              kwargs...))
    end

    return episodes
end

function Q_learning_episode!(Q, env; ϵ=0.10, γ=0.99, α=0.2)
    start = time()

    function policy(s, bool)
        if rand() < ϵ && bool
            return rand(actions(env))
        else
            return argmax(a->Q[(s, a)], actions(env))
        end
    end

    s = observe(env)
    hist = [s]
    a = policy(s, true)
    r = act!(env, a)

    while !terminated(env)
        a = policy(s, true)
        r = act!(env, a)
        sp = observe(env)

        ap = policy(sp, false)

        Q[(s,a)] += α*(r + γ*Q[(sp, ap)] - Q[(s, a)])

        s = sp
        push!(hist, sp)
    end
```

```
    Q[(s,a)] += α*(r - Q[(s, a)])

    return (hist=hist, Q = copy(Q), time=time()-start)
end

function Q_learning!(env; n_episodes=100_000)
    Q = Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    episodes = []

    for i in 1:n_episodes
        reset!(env)
        push!(episodes, Q_learning_episode!(Q, env;
                                        ϵ=max(0.1, 1-i/n_episodes)))
    end

    return episodes
end

function double_Q_learning_episode!(Q1, Q2, N, env; ϵ=0.10, γ=0.99,
α=0.2, c=100)
    start = time()

    function policy(s)
        bonus(nsa, ns) = nsa == 0 ? Inf : sqrt(log(ns)/nsa)
        Ns = sum(N[(s,a)] for a in actions(m))
        return argmax(a->Q1[(s,a)] + c*bonus(N[(s,a)], Ns),
actions(m))
    end

    A = collect(actions(env))
    s = observe(env)
    hist = [s]
    a = policy(s)
    r = act!(env, a)

    while !terminated(env)
        a = policy(s)
        r = act!(env, a)
        sp = observe(env)

        ap = policy(sp)

        N[(s, a)] = get(N, (s, a), 0.0) + 1

        Q1[(s,a)] += α*(r + γ*Q2[(sp, A[argmax(Q1[(sp,ap)] for ap in
A)])] - Q1[(s, a)])
        Q2[(s,a)] += α*(r + γ*Q1[(sp, A[argmax(Q1[(sp,ap)] for ap in
A)])] - Q2[(s, a)])
```

```
        s = sp
        push!(hist, sp)
    end

    Q1[(s,a)] += α*(r - Q2[(s, a)])
    Q2[(s,a)] += α*(r - Q1[(s, a)])

    return (hist=hist, Q = copy(Q1), time=time()-start)
end

function double_Q_learning!(env; n_episodes=100_000)
    Q1 = Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    Q2 = Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    N =  Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    episodes = []

    for i in 1:n_episodes
        reset!(env)
        push!(episodes, double_Q_learning_episode!(Q1, Q2, N, env;
                                        ϵ=max(0.1, 1-i/n_episodes)))
    end

    return episodes
end

m = gw
env = convert(AbstractEnv, m)

using Plots

function evaluate(env, policy, n_episodes=100_000, max_steps=1000,
γ=1.0)
    returns = Float64[]
    for _ in 1:n_episodes
        t = 0
        r = 0.0
        reset!(env)
        s = observe(env)
        while !terminated(env)
            a = policy(s)
            r += γ^t*act!(env, a)
            s = observe(env)
            t += 1
        end
        push!(returns, r)
    end
```

```
    return returns
end

sarsa_episodes             = sarsa!(            env,
n_episodes=100_000);
Q_learning_episodes        = Q_learning!(       env,
n_episodes=100_000);
# double_Q_learning_episodes = double_Q_learning!(env,
n_episodes=100_000);
# lambda_episodes            = sarsa_lambda!(     env,
n_episodes=100_000, α=0.1, λ=0.3);

# render(env, color = s -> maximum(map(a-
>last(sarsa_episodes).Q[(s,a)],actions(env))))
# render(env, color = s -> maximum(map(a-
>last(Q_learning_episodes).Q[(s,a)],actions(env))))

episodes = Dict("Q_learning"=>Q_learning_episodes,
"SARSA"=>sarsa_episodes)

p1 = plot(xlabel="steps in environment", ylabel="avg return")
n = 2_000
stop = 100_000
for (name, eps) in episodes
    Q = Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    xs = [0]
    ys = [mean(evaluate(env, s->argmax(a->Q[(s, a)], actions(env))))]
    for i in n:n:min(stop, length(eps))
        newsteps = sum(length(ep.hist) for ep in eps[i-n+1:i])
        push!(xs, last(xs) + newsteps)
        Q = eps[i].Q
        push!(ys, mean(evaluate(env, s->argmax(a->Q[(s, a)],
actions(env)))))
    end
    plot!(p1, xs, ys, label=name)
    lastval = last(ys)
    display("$name $lastval")
end
p1
display(p1)

p2 = plot(xlabel="wall clock time", ylabel="avg return")
n = 2_000
stop = 100_000
for (name,eps) in episodes
    Q = Dict((s, a) => 0.0 for s in observations(env), a in
actions(env))
    xs = [0.0]
    ys = [mean(evaluate(env, s->argmax(a->Q[(s, a)], actions(env))))]
```

```
    for i in n:n:min(stop, length(eps))
        newtime = sum(ep.time for ep in eps[i-n+1:i])
        push!(xs, last(xs) + newtime)
        Q = eps[i].Q
        push!(ys, mean(evaluate(env, s->argmax(a->Q[(s, a)],
actions(env)))))
    end
    plot!(p2, xs, ys, label=name)
end
p2
```