<A>**Online supplement**

**[INSERT ONLINE SUPPLEMENT from separate file]**

<A>**MATLAB code**

This section contains the different MATLAB files used to simulate the forager and the environment described in the article. In particular, the first one (startAgent.m) provides the different variables whose numerical values could be modified, and the second one (updateInnerStates.m) notably includes the equations governing the forager’s food-seeking behavior. Note that the code has been modified since its initial creation, with some symbols referring to notions no longer used thereafter. Also, mass-dependent predation risk (*R*mass in the description above) was described in terms of mass-dependent safeness (*S*m) relative to predators. This unconventional formulation was therefore avoided in the article. In the code, *S*m = *R*mass.

**File 1: startAgent.m**

%% Init experiment

rng('shuffle');

% Experimental variables

trial.plot = 0; % Set to 1 to plot the movement of the bot

trial.text = 1; % Set to 1 to write some output to the command window

trial.save = 1; % Set to 1 to save results to hard drive

trial.storeSteps = 50; % Store inner states of the bot every given number of steps

% (in addition to the CS/UCS save)

trial.endProgram = 25000; % Number of steps required to stop the program

% Field variables

trial.rows = 500; % Rows of field

trial.cols = 500; % columnds of field

trial.numCS = 0; % Number of CS on the field: CS --> 1

trial.numUCS = 0; % Number of UCS on the field: UCS --> 2

trial.optDen = 800; % Optimal density!

% Definitions for the agent (increments and decrements)

trial.EDecrease = 0.001; % Decrease of energy for each step

trial.EIncrease = 0.25; % Increase of energy after finding an UCS

trial.EShort2Long = 0.004; % Energy that moves from short to long-term battery during % each step

trial.E\_Sm = 50; % Threshold for computation of mass-dependent saveness

trial.ERestSteps = 400; % Number of steps to be done after the probability of % resting is checked

trial.ERestDecrease = 0.09; % Decreasse of long term energy during resting

trial.moveProbMax = 0.95; % Maximum value that can be reached to walk into a % direction of a CS/UCS or away from the border

trial.moveProbDecr = 0.0005; % Decrement of probability for each step

trial.memLen = 30; % "Entries" in memory that are used to compute the subjective probability

trial.memUCSLen = 30; % Entries for the number of steps walked between two UCS

trial.memCSLen = 20; % Entries for the number of steps walked between two CS

trial.memCSPosLen = 5; % Entries for the coordinates of the last CS the bot found

trial.ignoreUCS = 5; % Number of steps where the bot ignores any CS or UCS % after finding a UCS (set 0 to disable)

trial.ignoreCS = 1; % Number of steps where the bot ignores any CS or UCS % after finding a CS (set 0 to disable)

% Initial values for the agent

trial.initELong = 5; % Initial energy of the long-term battery 0 to 10)

trial.initWh = 0; % Initial hope related wanting (0 to 0.25)

trial.initWn = 0.5; % Initial need ralated wanting (0 to 1)

trial.initSe = 1; % Environmental saveness (0 to 1) is a fixed value that % doesn't change

trial.initSm = 0.3; % Mass-dependent saveness (0 to 1) is a varying value % (depends on long term energy)

trial.initCSDetect = 3; % Initial value for CS detect

trial.initV = [1 1]; % Stress vulnerability (a values between the given two will % be choosen)

% Conditions to end the game

trial.satiety = 10; % Threshold that defines the satiety to leave the % patch (energy of long-term battery high enough)

trial.exhaustion = 0; % Threshold that defines when the energy level of the % long-term battery is too low

% Input check

if(trial.numUCS > trial.optDen)

error('trial.numUCS > trial.optDen is not allowed !!!');

end

%% Init bot

moveDir = randperm(4);

moveDir = moveDir(1);

bot.x = round(trial.cols/2); % x position in field

bot.y = round(trial.rows/2); % y position in field

bot.E = 0; % Energy of the short-term battery (0 to 1)

bot.ELong = trial.initELong; % Energy of long-erm battery (0 to 10)

bot.moveP = [0.25 0.25 0.25 0.25]; % Search probability for all directions

bot.moveDir = moveDir; % Actual moving direction of bot

bot.moveDirOld = 0; % the moving direction before

bot.CSDetect = trial.initCSDetect; % Distance between bot and CS to % rise probability to move towards the CS (computed % in updateInnerStates)

bot.Wn = trial.initWn; % Need related wanting -> Motivation to seek food (0 to 1)

bot.memory = 0; % Sequence of trials (CS=0 and UCS=1) as average

bot.memCSx = zeros(1,trial.memCSPosLen); % Memory of last CS (x pos)

bot.memCSy = zeros(1,trial.memCSPosLen); % Memory of last CS (y pos)

bot.memUCS = zeros(1,trial.memCSLen); % Memory of number of steps between the UCS

bot.memUCSTot = zeros(1,trial.memUCSLen); % Memory of total number of steps for every USC

bot.memCS = zeros(1,trial.memCSLen); % Memory of number of steps between the CS

bot.memCSTot = zeros(1,trial.memCSLen); % Memory of total number of steps for every CS

bot.countUCS = 0; % Counter of the UCS

bot.countCS = 0; % Counter of the CS

bot.countCSPos = 0; % Counter of the last visited CS potitions

bot.p = 0; % Subjective probability

bot.u = 0; % Reward uncertainty

bot.Wh = trial.initWh; % Hope related wanting (0 to 0.25)

bot.Wt = 0; % Total wanting (0 to some computed value)

bot.c = 0; % Coefficient of variation

bot.a = 0; % Decrease in attractiveness

bot.Se = trial.initSe; % Environmental saveness

bot.Sm = trial.initSm; % Mass-dependent saveness

bot.S = 1/(trial.numUCS/trial.optDen)-1; % Multiplicative factor which % corresponds to "expected scarcity"

bot.v = (trial.initV(2) - trial.initV(1)) \* rand + trial.initV(1); % Stress % vulnerability

bot.pRest = 0; % Probability to res after number of steps

bot.trial = 0; % Counter for the number of trials the bot did

bot.alive = 1;

bot.trial = 0; % Number of trials, trial counter

bot.stepCount = 0; % counts the number of loops/steps

bot.endGame = 0; % Indicator to end the game

bot.directCounter = [0 0 0 0]; % Counter that for the direction over the whole experiment

bot.directProb = bot.Wt/1.5; % Probability for a special direction if bot is % closer than X to a CS/UCS (a function of % Wt --> computed in updateInnerStates.m)

bot.close2CSUCS = 0; % An indicator if the bot is close to a CS/UCS

bot.close2Border = 0; % An indicator if the bot is close to a border or not

bot.counterSave = 0; % Counter for saving the data

bot.result = 0; % Field that is used to show if bot found a CS or UCS

bot.resting = 0; % Variable is set to one if the bot rests

bot.ignoreUCSCount = 0; % Counter that counts for how long we ignore any % CS/UCS after finding a UCS

bot.ignoreCSCount = 0; % Counter that counts for how long we ignore any % CS/UCS after finding a CS

%% Init environment

if(trial.plot == 1)

trial.text = 1;

end

trial.env = zeros(trial.rows, trial.cols);

% First CS: on this field the value changes to 1

% Get random numbers ranging from 1 to rows\*cols

tmp = randperm(trial.rows\*trial.cols, trial.numCS);

trial.env(tmp) = 1;

% Now UCS: on this field the value changes to 2

trial.env = addUCS2Field(trial.env, trial.numUCS);

%% Set up area to plot

if(trial.plot == 1)

figure;

set(gca, 'Xlim', [0 trial.cols], 'Ylim', [0 trial.rows])

% Plot CS and UCS

[rCS,cCS] = ind2sub([trial.rows, trial.cols], find(trial.env==1));

[rUCS,cUCS] = ind2sub([trial.rows, trial.cols], find(trial.env==2));

hold on

plot(cUCS,rUCS, '.g');

plot(cCS, rCS, '.b');

pind = plot(bot.x, bot.y,'r\*');

hold off

drawnow

end

%% Start main loop

i = 0;

k = 0;

l = 0;

m = 0;

try

while(i <= trial.endProgram)

i=i+1;

%% Plot bot

if(trial.plot == 1)

set(pind, 'XData', bot.x, 'YData', bot.y)

drawnow

grid on

end

%% Move to next position

[trial, bot, newDirection] = moveNextStep(trial, bot);

%% Update inner states

[bot, newTrial] = updateInnerStates(trial, bot);

%% Save data for analysis

if(mod(bot.stepCount, trial.storeSteps) == 0)

% Save data for this step as well

k = k+1;

saveBot(k) = bot;

% Compute mean of saveBot

l = l+1;

meanBot(l) = computeMean(saveBot);

% Clear counter

k = 0;

else

% Save data

k = k+1;

saveBot(k) = bot;

end

% Save data for every new direction the bot chose

if(newDirection == 1)

m = m+1;

newDirBot(m) = bot;

end

% Save data for each step (only some information)

rawBot(i).x = bot.x;

rawBot(i).y = bot.y;

rawBot(i).moveDir = bot.moveDir;

rawBot(i).trial = bot.trial;

rawBot(i).resting = bot.resting;

rawBot(i).countCS = bot.countCS;

rawBot(i).countUCS = bot.countUCS;

rawBot(i).result = bot.result;

%% Check if we have to update the environment

% After consuming an UCS it's associated CS disappears and a new UCS occurs

% elsewhere in the environment

if(newTrial > 0 || mod(bot.stepCount, trial.storeSteps) == 0)

% Save state of bot for every trial after first CS/UCS

bot.counterSave = bot.counterSave+1;

agent(bot.counterSave) = bot;

% Prepare data for next trial and plot some user info

if(newTrial == 1 && trial.text == 1)

disp([num2str(i) '/' num2str(bot.trial) ' - Found CS (' num2str(agent(bot.counterSave).countCS) ' in total) - New trial'])

elseif(newTrial == 2)

trial.env(bot.y, bot.x) = 0;

trial.env = addUCS2Field(trial.env, 1);

if(trial.text == 1)

disp([num2str(i) '/' num2str(bot.trial) ' - Found UCS (' num2str(agent(bot.counterSave).countUCS) ' in total) - New trial'])

end

elseif(newTrial == 3 && trial.text == 1)

% User info

disp([num2str(bot.stepCount) ' - Energy decrease of ' num2str(trial.ERestDecrease) ' - RESTING']);

elseif(newTrial == 99 && trial.text == 1)

disp([num2str(i) '/' num2str(bot.trial) ' - Last entry!'])

end

end

%% Check when the game ends

if(bot.endGame > 0)

i = inf;

end

end

hold off

disp('DONE !!!');

disp(['Number of UCS: ' num2str(bot.countUCS)]);

disp(['Number of CS: ' num2str(bot.countCS)]);

% More user info

computeRestPhase;

%% Plot results

figure

plotResult(agent)

%% Save data

if(trial.save == 1)

save(['.\Result\' datestr(now,'YYYYmmDDHHMM') '\_solitaryForaging2.mat'], 'trial', 'bot', 'agent', 'meanBot', 'saveBot', 'rawBot', 'newDirBot');

end

catch

% Save what we have

tmp = datestr(now, 'YYYYmmDDHHMMSS');

err = lasterror;

save(['ErrorSave' tmp]);

disp('ERROR !!!')

end

**File 2: updateInnerState.m**

%% Init variables

newTrial = 0;

actField = trial.env(bot.y, bot.x);

%% Handle inner states depending on position

% actField = 0 --> no CS and no UCS

% actField = 1 --> CS but no UCS (hole, odor, noise)

% actField = 2 --> UCS (food)

if(actField ~= 0)

%% Update trial counter

bot.trial = bot.trial+1;

index = mod(bot.trial-1,trial.memLen)+1;

%% Set variables specific for CS and UCS

if(actField == 1)

% Set new trial to 1 to signal that we found a CS

% -----------------------------------------------

newTrial = 1;

% Memory

bot.memory(index) = 0;

bot.result = 1;

% Save position of last CS (UCS fanishes anyway)

bot.countCSPos = bot.countCSPos+1;

indexCSPos = mod(bot.countCSPos, trial.memCSPosLen)+1;

bot.memCSx(indexCSPos) = bot.x;

bot.memCSy(indexCSPos) = bot.y;

% Save distance between this and last CS

indexCS = mod(bot.countCS, trial.memCSLen)+1;

indexCSLast = mod(indexCS-2, trial.memCSLen)+1; % Entry of CS before

bot.memCS(indexCS) = bot.stepCount - bot.memCSTot(indexCSLast);

bot.memCSTot(indexCS) = bot.stepCount;

bot.countCS = bot.countCS + 1;

% Update field to ignore the new CS/UCS for the next nunmber of steps

bot.ignoreCSCount = trial.ignoreCS;

elseif(actField == 2)

% Set new trial to 2 to signal that we found a UCS

% ------------------------------------------------

newTrial = 2;

% Memory

bot.memory(index) = 1;

bot.result = 2;

% Save distance between this and last CS

indexUCS = mod(bot.countUCS, trial.memUCSLen)+1;

indexUCSLast = mod(indexUCS-2, trial.memUCSLen)+1; % Entry of CS before

bot.memUCS(indexUCS) = bot.stepCount - bot.memUCSTot(indexUCSLast);

bot.memUCSTot(indexUCS) = bot.stepCount;

bot.countUCS = bot.countUCS + 1;

% Compute coefficient of variation

bot.c = std(bot.memUCS)/mean(bot.memUCS);

% Compute decrease in attractiveness

bot.a = 1/(1+exp(-0.05\*mean(bot.memUCS)+5));

% Update field to ignore the new CS/UCS for the next nunmber of steps

bot.ignoreUCSCount = trial.ignoreUCS;

end

%% Now update general CS and UCS entries

% Subjective probability of reward

bot.p = sum(bot.memory)/length(bot.memory);

% Reward uncertainty

bot.u = bot.p\*(1-bot.p);

% Hope induced wanting

bot.Wh = bot.Wn\*(bot.p\*(1-bot.p) + bot.S\*bot.c);

end

%% Compute general inner states for each step

bot = energyUsage(bot, trial, actField);

% Need related wanting

% Compute Wn as a nonlinear function of the long term energy.

% Wn ranges from 0 to 1. Long term energy ranges from 0 to 10

bot.Wn = 1-(10^-4\*bot.ELong^4);

if(bot.Wn < 0)

bot.Wn = 0;

end

% Compute Mass-dependent saveness

% Sm depends on the long-term energy: Sm gradually decreases when energy

% overcomes a fixed threshold according to an exponential curve)

%bot.Sm = 1-(10^-trial.E\_Sm \* bot.ELong^trial.E\_Sm);

bot.Sm = 0.15\*bot.ELong - 0.5;

% Condition unter which predation risk is cencelled

if(bot.ELong <= 3.5)

bot.Sm = 0;

end

% Finally compute total Wanting

% bot.Wt = bot.Se\*bot.Sm\*(bot.Wn+bot.Wh);

bot.Wt = bot.Wn+bot.Wh;

% Adjust CSDetect depending on Wt

bot.CSDetect = round(12/(1+exp(-4\*bot.Wt+4)));

% Compute the probability to walk into the direction of a CS/USC. The

% probability depends on Wt and is linear in the range of Wt=0 to Wt=1.5.

% Because Wt could theoretically be infinitive the probability is

% set to a fixed maximum value.

if(bot.Wt >= 0 && bot.Wt <= 1.5)

bot.directProb = (bot.Wt/1.5) \* trial.moveProbMax;

elseif(bot.Wt > 1.5)

bot.directProb = trial.moveProbMax;

elseif(bot.Wt < 0)

bot.directProb = 0;

end

% Compute probability to rest after a given number of steps

%bot.pRest = (1-bot.Se\*bot.Sm + trial.numUCS/trial.optDen)/2;

bot.pRest = (bot.Se\*bot.Sm + trial.numUCS/trial.optDen)/2;

if(bot.pRest > 1)

bot.pRest = 1;

end

%% Check for resting

% Check resting only after given amount of steps!

bot.resting = 0;

if(mod(bot.stepCount, trial.ERestSteps) == 0)

num1 = round(100\*bot.pRest);

num0 = 100-num1;

num = [ones(1,num1), zeros(1,num0)];

num(randperm(100)) = num;

if(num(1) == 1)

% Set new trial to 3 to signal that rest

% --------------------------------------

newTrial = 3;

bot.resting = 1;

if((bot.ELong - trial.ERestDecrease) >= 0)

bot.ELong = bot.ELong - trial.ERestDecrease;

else

bot.ELong = 0;

end

end

end

%% Check, if the game continues

if(bot.ELong >= trial.satiety)

disp('Threshold to leave the patch is reached! - Satiety is high enough!');

bot.endGame = 2;

end

if(bot.ELong <= trial.exhaustion)

disp('Energylevel too low. Bot is dead :(');

bot.alive = 0;

bot.endGame = 3;

newTrial = 99;

end

if(bot.endGame > 0)

bot.trial = bot.trial + 1;

newTrial = 99;

end

**File 3: moveNextStep.m**

%% Init variables

x = bot.x;

y = bot.y;

moveP = bot.moveP;

newDirection = 0;

%% Check if bot is close the border of the field

[moveDir, bot] = check4Borders(bot,trial);

%% Choose new direction

% Check if direction is close to a CS or UCS and move to that direction.

% Choose a random direction if there is nothing around

if(bot.close2Border == 0)

[moveDir, bot] = findDirection2Target(bot, trial);

end

% Set probability high for given direction to walk to a CS/UCS

if(bot.close2CSUCS > 0)

if(moveDir == 1)

% Set probability high to direction of CS/USC

moveP = [bot.directProb (1-bot.directProb)/3 (1-bot.directProb)/3 (1-bot.directProb)/3];

elseif(moveDir == 2)

% Set probability high to direction of CS/USC

moveP = [(1-bot.directProb)/3 bot.directProb (1-bot.directProb)/3 (1-bot.directProb)/3];

elseif(moveDir == 3)

% Set probability high to direction of CS/USC

moveP = [(1-bot.directProb)/3 (1-bot.directProb)/3 bot.directProb (1-bot.directProb)/3];

elseif(moveDir == 4)

% Set probability high to direction of CS/USC

moveP = [(1-bot.directProb)/3 (1-bot.directProb)/3 (1-bot.directProb)/3 bot.directProb];

end

end

% Set probability even higher if we are close to a border! We want to go

% away from that border !!!

if(bot.close2Border > 0)

if(moveDir == 1)

% Set probability high to direction of CS/USC

moveP = [trial.moveProbMax (1-trial.moveProbMax)/3 (1-trial.moveProbMax)/3 (1-trial.moveProbMax)/3];

elseif(moveDir == 2)

% Set probability high to direction of CS/USC

moveP = [(1-trial.moveProbMax)/3 trial.moveProbMax (1-trial.moveProbMax)/3 (1-trial.moveProbMax)/3];

elseif(moveDir == 3)

% Set probability high to direction of CS/USC

moveP = [(1-trial.moveProbMax)/3 (1-trial.moveProbMax)/3 trial.moveProbMax (1-trial.moveProbMax)/3];

elseif(moveDir == 4)

% Set probability high to direction of CS/USC

moveP = [(1-trial.moveProbMax)/3 (1-trial.moveProbMax)/3 (1-trial.moveProbMax)/3 trial.moveProbMax];

end

end

%% MOVE BOT

% Set next step

[x, y] = moveBot(x, y, moveDir, trial);

%% Decrease direction probability

% Positions in direction vector [forward right backward left]

% 1 --> forward | 2 --> right | 3 --> backward | 4 --> left

if(moveDir == bot.moveDir)

% Bot moves into the same direction --> decrement probability and increment the other prob values

if(moveDir == 1)

moveP = [bot.directProb-trial.moveProbDecr (trial.moveProbDecr+1-bot.directProb)/3 (trial.moveProbDecr+1-bot.directProb)/3 (trial.moveProbDecr+1-bot.directProb)/3];

elseif((bot.moveDir == 2))

moveP = [(trial.moveProbDecr+1-bot.directProb)/3 bot.directProb-trial.moveProbDecr (trial.moveProbDecr+1-bot.directProb)/3 (trial.moveProbDecr+1-bot.directProb)/3];

elseif((bot.moveDir == 3))

moveP = [(trial.moveProbDecr+1-bot.directProb)/3 (trial.moveProbDecr+1-bot.directProb)/3 bot.directProb-trial.moveProbDecr (trial.moveProbDecr+1-bot.directProb)/3];

elseif((bot.moveDir == 4))

moveP = [(trial.moveProbDecr+1-bot.directProb)/3 (trial.moveProbDecr+1-bot.directProb)/3 (trial.moveProbDecr+1-bot.directProb)/3 bot.directProb-trial.moveProbDecr];

end

else

% Directtion has changed! Set indicator to 1

newDirection = 1;

end

%% Copy coordinates to bot struct

bot.x = x;

bot.y = y;

% Save new direction

bot.moveDirOld = bot.moveDir;

bot.moveDir = moveDir;

% save new moving probability

bot.moveP = moveP;

%% Count steps

bot.stepCount = bot.stepCount+1;

bot.directCounter(bot.moveDir) = bot.directCounter(bot.moveDir)+1;

**File 4: moveBot.m**

%% Check for every direction

if(moveDir == 1)

if(y+1 <= trial.rows)

y = y+1;

end

elseif(moveDir == 2)

if(x+1 <= trial.cols)

x = x+1;

end

elseif(moveDir == 3)

if(y-1 > 0)

y = y-1;

end

elseif(moveDir == 4)

if(x-1 > 0)

x = x-1;

end

end

**File 5: getMovingDataFromBot.m**

%% Load data

[f,p] = uigetfile;

load([p, f]);

len = length(rawBot);

%% Init variables

if(nargin == 0)

startStep = 1;

stopStep = len;

elseif(nargin == 1)

stopStep = len;

end

x = [rawBot(startStep:stopStep).x];

y = [rawBot(startStep:stopStep).y];

moveDir = [rawBot(startStep:stopStep).moveDir];

%% Plot trajectories

figure;

plot(x, y);

% Set axes to the environment

set(gca, 'Xlim', [0 trial.cols], 'Ylim', [0 trial.rows])

grid on

% Plot CS and UCS

[rCS,cCS] = ind2sub([trial.rows, trial.cols], find(trial.env==1));

[rUCS,cUCS] = ind2sub([trial.rows, trial.cols], find(trial.env==2));

hold on

plot(cUCS,rUCS, '.g');

plot(cCS, rCS, '.b');

pind = plot(bot.x, bot.y,'r\*');

hold off

%% Compute steps on a straight line

i = 1;

j = 1;

cStep = moveDir(1);

bSteps = 0; % Number of steps the bot walks into one direction

bAngle = 0; % Change of angle for each new direction

bDirect = 0; % Direction where the bot moves

bStepPos = 0; % Number of steps when a change occurs

while(i<length(moveDir))

% Does bot move into the same direction?

if(moveDir(i) == cStep)

% Count steps

bSteps(j) = bSteps(j)+1;

bDirect(j) = moveDir(i);

else

% Bot moves into new direction -> compute new angle

% 1 --> forward

% 2 --> right

% 3 --> backward

% 4 --> left

if(ismember(moveDir(i)-moveDir(i-1), [-3 1]))

bAngle(j+1) = 90;

elseif(ismember(moveDir(i)-moveDir(i-1), [-1 3]))

bAngle(j+1) = -90;

elseif(ismember(moveDir(i)-moveDir(i-1), [-2 2]))

bAngle(j+1) = 180;

else

disp('Error');

end

% Save step count when change occurs

bStepPos(j+1) = i+startStep-1;

% Increment event counter and init variables for new direction

j = j+1;

bSteps(j) = 0;

cStep = moveDir(i);

% Count steps

bSteps(j) = bSteps(j)+1;

bDirect(j) = moveDir(i);

end

% Increment i

i = i+1;

end

%% Save all in one matrix

res = [bSteps;bDirect;bAngle;bStepPos];

%% Save to matlab file

filename = [p f(1:end-4) '\_analysis.mat'];

save(filename, 'res', 'agent', 'trial', 'rawBot');

disp(['File saved: ' filename]);

**File 6: findDirection2Target.m**

%% Init variables

order = randperm(4);

i = 1;

out = 0;

ignoreFood = 0;

%% After consuming a CS/UCS the bot ignores all new CS/UCS for a certain number of steps

if(bot.ignoreUCSCount > 0)

ignoreFood = 1;

bot.ignoreUCSCount = bot.ignoreUCSCount-1;

end

if(bot.ignoreCSCount > 0)

ignoreFood = 1;

bot.ignoreCSCount = bot.ignoreCSCount-1;

end

%% Check environment

while(ignoreFood==0 && out == 0 && i<=length(order))

[out, bot] = checkEnvironment4Targets(bot, trial, order(i));

i = i+1;

end

%% Choose random direction

if(out == 0)

out = chooseRandomDirection(bot);

end

**File 7: findCSAndUCS.m**

%% Check upper right corner for CS/UCS

tmp=trial.env(bot.y+(0:CSDetect-1), bot.x+(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x+xxp(ind)-1;

yp = bot.y+yyp(ind)-1;

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x == xp && bot.y == yp)

disp('DONE')

trial.env(bot.y, bot.x) = 0;

else

if(angl < 45)

% Move to the right

disp('Right')

%bot.x = bot.x-1;

else

% Move upwards

disp('Up');

%bot.y = bot.y+1;

end

end

end

%%

tmp=trial.env(bot.y+(0:CSDetect-1), bot.x+(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

xp = bot.x+find(any(tmp),1)-1;

yp = bot.y+find(any(tmp'),1)-1;

angl = 180\*atan((yp-bot.y)/(xp-bot.x))/pi; % Angle between bot ans CS/UCS

if(bot.x == xp && bot.y == yp)

disp('DONE')

trial.env(bot.y, bot.x) = 0;

else

if(angl < 45)

% Move to the right

disp('Right')

else

% Move upwards

disp('Up');

end

end

end

%% Check upper left corner for CS/UCS

tmp=trial.env(bot.y+(0:CSDetect-1), bot.x-(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x-xxp(ind)+1;

yp = bot.y+yyp(ind)-1;

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x == xp && bot.y == yp)

disp('DONE')

trial.env(bot.y, bot.x) = 0;

else

if(angl < 45)

% Move to the right

disp('Left')

%bot.x = bot.x-1;

else

% Move upwards

disp('Up');

%bot.y = bot.y+1;

end

end

end

%% Check lower right corner

tmp=trial.env(bot.y-(0:CSDetect-1), bot.x+(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x+xxp(ind)-1;

yp = bot.y-yyp(ind)+1;

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x == xp && bot.y == yp)

disp('DONE')

trial.env(bot.y, bot.x) = 0;

else

if(angl < 45)

% Move to the right

disp('Right')

%bot.x = bot.x-1;

else

% Move upwards

disp('Down');

%bot.y = bot.y+1;

end

end

end

%% Check lower left corner - WORKS WITH MORE THAN ONE MARKER !!!

tmp=trial.env(bot.y-(0:CSDetect-1), bot.x-(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x-xxp(ind)+1;

yp = bot.y-yyp(ind)+1;

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x == xp && bot.y == yp)

disp('DONE')

trial.env(bot.y, bot.x) = 0;

else

if(angl < 45)

% Move to the right

disp('Left')

%bot.x = bot.x-1;

else

% Move upwards

disp('Down');

%bot.y = bot.y+1;

end

end

end

**File 8: findClosestTarget.m**

%% Check input

if(length(xxp) == 1)

out = 1;

else

% Compute distance

d = sqrt(xxp.^2+yyp.^2);

% Find index of smalles distance

[~, out] = min(d);

end

**File 9: energyUsage.m**

%% Increase energy of short-term battery in case we found food

if(actField == 2)

bot.E = bot.E+trial.EIncrease;

end

%% Moving enery from short-term to long-term battery

if((bot.E-trial.EShort2Long) >= 0)

bot.ELong = bot.ELong+trial.EShort2Long;

bot.E = bot.E-trial.EShort2Long;

else

bot.ELong = bot.ELong+bot.E;

bot.E = 0;

end

%% Decrease energy of long-term battery for each step

bot.ELong = bot.ELong-trial.EDecrease;

if(bot.ELong < 0)

bot.ELong = 0;

end

**File 10: computeRestPhase.m**

%% Init variables

% Get index of rest trials

ind = logical([agent.resting]);

% Compute number of rest phases

numSteps = sum(ind);

% Compute distance between rest phases

diffSteps = diff([agent(ind).stepCount]);

%% User info

disp('-----------------------------------------------------------------');

disp([' Number of rest phases in total: ' num2str(numSteps)]);

disp([' Trialnumber of rest pase: ' num2str(find(ind))]);

disp([' Number of steps bewteen rest phases: ' num2str(diffSteps)]);

**File 11: choseRandomDirection.m**

%% Check direction depending on the inputs

if(nargin == 1)

% Coose random direction

p = round(bot.moveP\*100);

direction = [ones(1,p(1)) 2\*ones(1,p(2)) 3\*ones(1,p(3)) 4\*ones(1,p(4))];

direction = direction(randperm(length(direction)));

out = direction(1);

else

% Move away from the border!

if(ignoreDir == 1)

out = 3;

elseif(ignoreDir == 2)

out = 4;

elseif(ignoreDir == 3)

out = 1;

elseif(ignoreDir == 4)

out = 2;

end

end

**File 12: checkEnvironment4Targets.m**

%% Init variables

out = 0;

CSDetect = bot.CSDetect;

memX = bot.memCSx;

memY = bot.memCSy;

dbOut = 0;

%% Check upper right corner for CS/UCS

if(lrud == 1)

tmp=trial.env(bot.y+(0:CSDetect-1), bot.x+(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x+xxp(ind)-1;

yp = bot.y+yyp(ind)-1;

if(any(xp == memX) && any(yp == memY))

% We are close to an old CS/UCS --> move away!

out = round(rand+3); % 3 or 4

bot.close2CSUCS = 1;

else

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x ~= xp || bot.y ~= yp)

if(angl < 45)

% Move to the right

if(dbOut == 1), disp('Right'); end

out = 2;

bot.close2CSUCS = 1;

else

% Move upwards

if(dbOut == 1), disp('Up'); end

out = 1;

bot.close2CSUCS = 1;

end

end

end

end

elseif(lrud == 2)

tmp=trial.env(bot.y+(0:CSDetect-1), bot.x-(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x-xxp(ind)+1;

yp = bot.y+yyp(ind)-1;

if(any(xp == memX) && any(yp == memY))

% We are close to an old CS/UCS --> move away!

out = round(rand+2); % 2 or 3

bot.close2CSUCS = 1;

else

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x ~= xp || bot.y ~= yp)

if(angl < 45)

% Move to the right

if(dbOut == 1), disp('Left'); end

out = 4;

bot.close2CSUCS = 1;

else

% Move upwards

if(dbOut == 1), disp('Up'); end

out = 1;

bot.close2CSUCS = 1;

end

end

end

end

elseif(lrud == 3)

tmp=trial.env(bot.y-(0:CSDetect-1), bot.x+(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x+xxp(ind)-1;

yp = bot.y-yyp(ind)+1;

if(any(xp == memX) && any(yp == memY))

% We are close to an old CS/UCS --> move away!

if(rand > 0.5)% 1 or 4

out = 1;

else

out = 4;

end

bot.close2CSUCS = 1;

else

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x ~= xp || bot.y ~= yp)

if(angl < 45)

% Move to the right

if(dbOut == 1), disp('Right'); end

out = 2;

bot.close2CSUCS = 1;

else

% Move upwards

if(dbOut == 1), disp('Down'); end

out = 3;

bot.close2CSUCS = 1;

end

end

end

end

elseif(lrud == 4)

tmp=trial.env(bot.y-(0:CSDetect-1), bot.x-(0:CSDetect-1));

if(any(any(tmp)))

% Find x and y pos of UCS

[yyp,xxp]=ind2sub(size(tmp),find(tmp>0));

% Find closest CS/UCS

ind = findClosestTarget(xxp, yyp);

xp = bot.x-xxp(ind)+1;

yp = bot.y-yyp(ind)+1;

if(any(xp == memX) && any(yp == memY))

% We are close to an old CS/UCS --> move away!

out = round(rand+1); % 1 or 2

bot.close2CSUCS = 1;

else

% Compute angle

angl = abs(180\*atan((yp-bot.y)/(xp-bot.x))/pi); % Angle between bot ans CS/UCS

if(bot.x ~= xp || bot.y ~= yp)

if(angl < 45)

% Move to the right

if(dbOut == 1), disp('Left'); end

out = 4;

bot.close2CSUCS = 1;

else

% Move upwards

if(dbOut == 1), disp('Down'); end

out = 3;

bot.close2CSUCS = 1;

end

end

end

end

end

**File 13: check4Borders.m**

%% Init variables

x = bot.x;

y = bot.y;

CSDetect = bot.CSDetect;

moveDir = bot.moveDir;

%% Check for borders

if(bot.close2Border == 0)

if(~(x+CSDetect-1 <= trial.cols && x-CSDetect-1 >= 1 && y+CSDetect-1 <= trial.cols && y-CSDetect-1 >= 1))

moveDir = chooseRandomDirection(bot, bot.moveDir);

bot.close2Border = 4\*bot.CSDetect;

end

else

bot.close2Border = bot.close2Border-1;

end

**File 14: addUCS2Field.m**

%% Get random positions for UCS

% Let's create ten times more positions than we need, in case a position is

% blocked

tmp = randperm(size(field,1) \* size(field,2), numberOfUCS\*10);

%field(tmp(1:numberOfUCS))

%% Find position where a UCS or CS exists and delete from index vector

while(~all(field(tmp(1:numberOfUCS)) == 0))

% Find positions where an UCS or CS exists

index = field(tmp(1:numberOfUCS)) > 0;

% Delete from tmp index vector

tmp(index) = [];

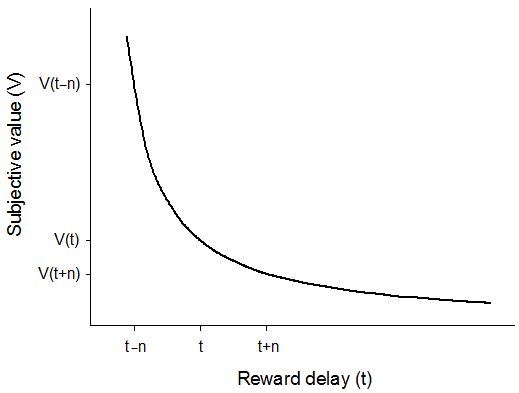
disp('One position for UCS was occupied');

end

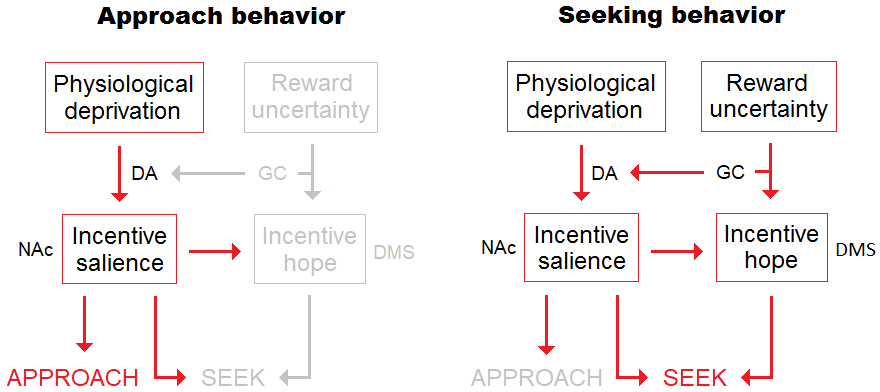
%% Copy UCS into field

field(tmp(1:numberOfUCS)) = 2;

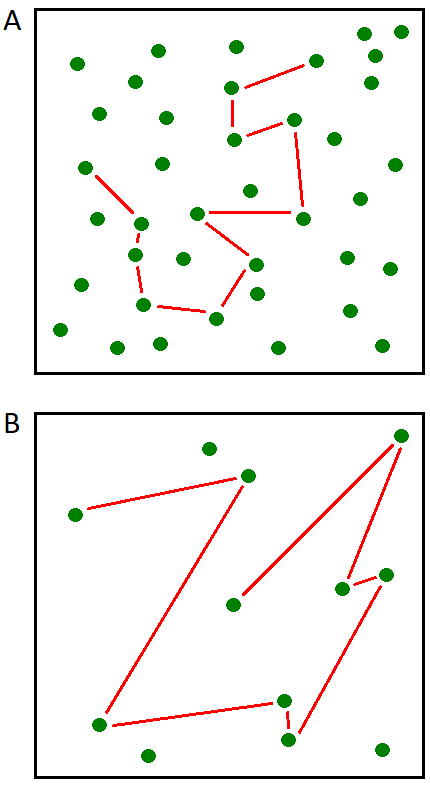
**Figures**



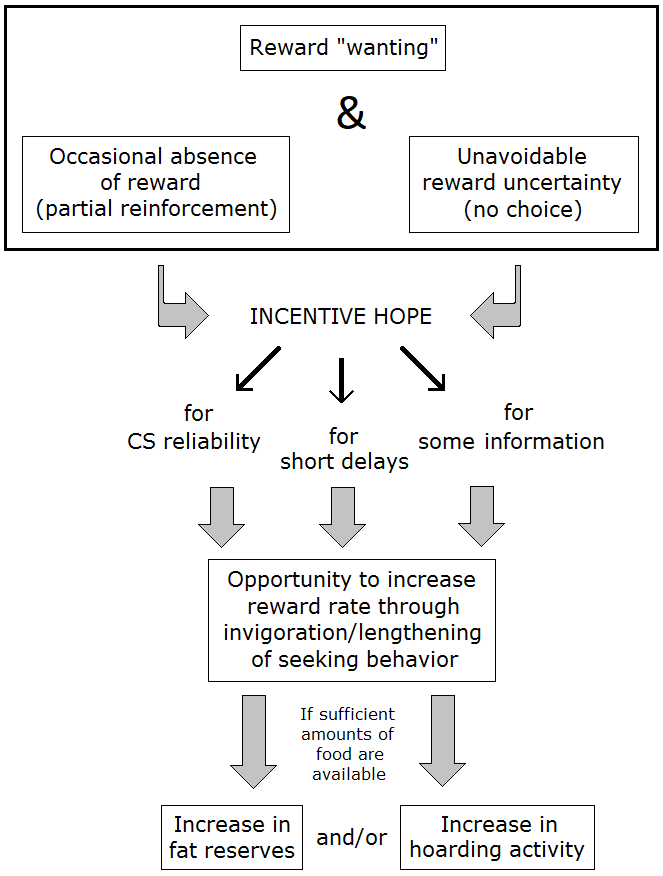
<CAPT>Fig. 1. The subjective value *V* of a delayed reward decreases in a hyperbolic fashion as the delay (or time *t*) before receiving that reward increases. As a result, a variable-delay schedule (reward after *t* – *n* or *t* + *n*) is preferred over a constant-delay schedule (reward after *t*) equal to its mean. Here, *V*(*t*) is smaller than the mean subjective value between *V*(*t* + *n*) and *V*(*t* – *n*) – a property called Jensen’s inequality – because of the high attractiveness of immediate or rapid rewards (received at *t* – *n*) in comparison with more delayed rewards (received at *t* + *n*).



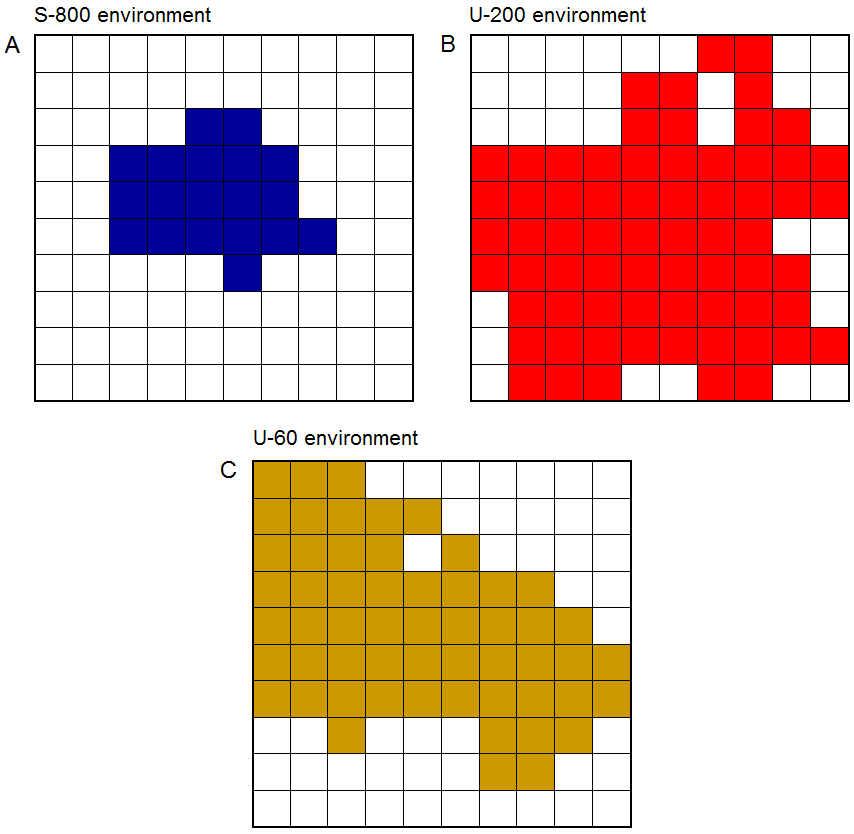
<CAPT>Fig. 2. Interaction between incentive salience (“wanting”) and incentive hope. The left side depicts a situation where no uncertainty is present (the food items are predictable or accessible). Approach behavior results from incentive salience processes (involving the release of dopamine in the nucleus accumbens), while seeking behavior remains subactivated and, hence, prevented. The right side represents what happens when the animal is subject to food uncertainty (unpredictable access). The animal comes to produce not only incentive salience but also incentive hope. Although approach could potentially be produced, its expression is canceled to the detriment of seeking, because the former behavior receives less activation than the latter. Of course, if uncertainty is temporarily abolished, then seeking is prevented and the stimulus is approached. This simple schema suggests that approach and seeking are differently processed while depending on the same motivational basis. It shows how a change in reward uncertainty can mechanically convert approach into seeking, and vice versa. DA = dopamine, GC = glucocorticoids, NAc = nucleus accumbens, DMS = dorsomedial striatum.



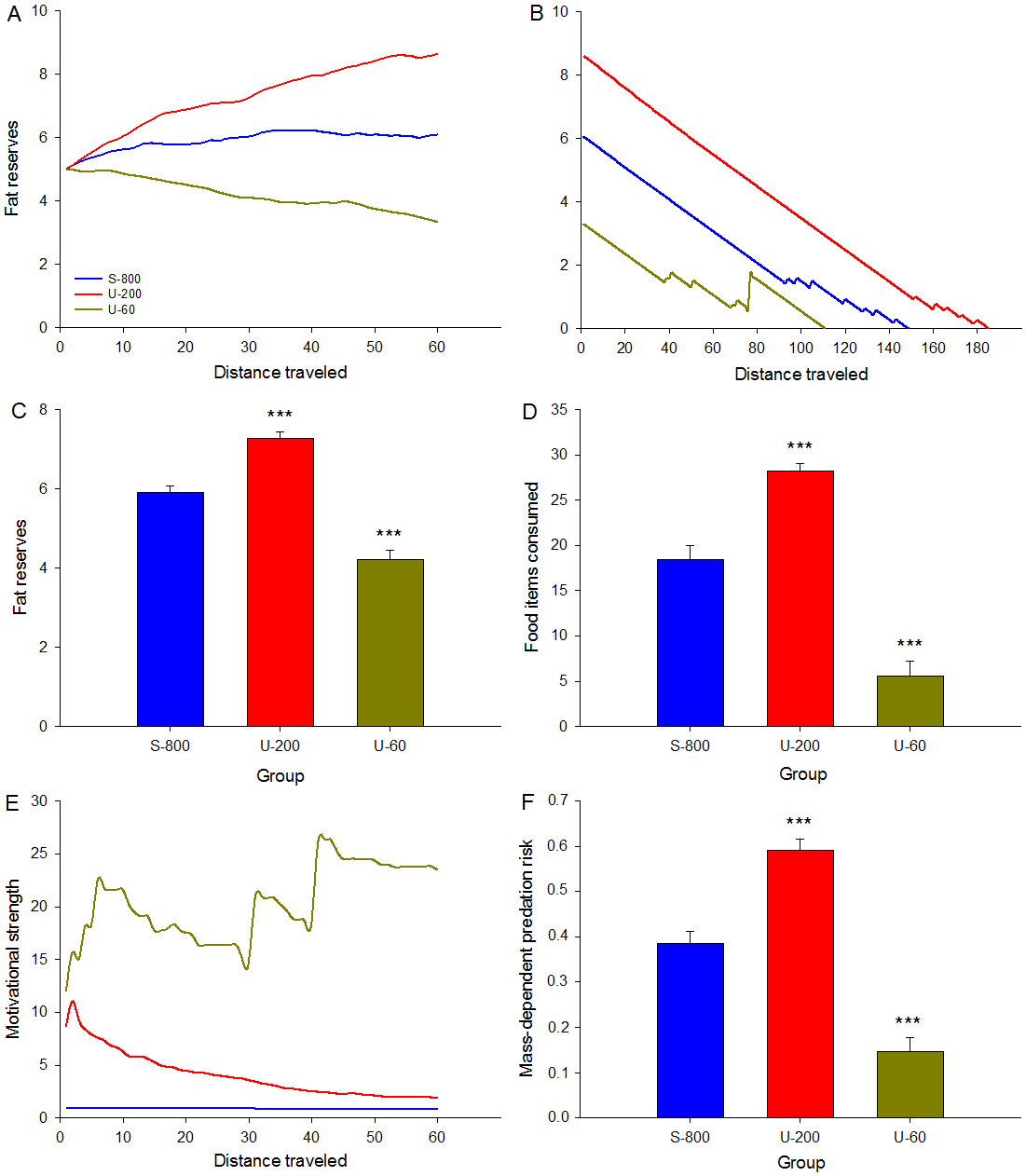
<CAPT>Fig. 3. (A) When food is abundant, there is no risk of starvation because the mean delay (the straight lines) to find edible items is short. (B) When food is scarce, there is a risk of starvation because the mean delay (the straight lines) to find edible items is longer. In such an environment, animals should hope for delays shorter than the mean and act accordingly.



<CAPT>Fig. 4. The incentive hope hypothesis. On top, the three conditions required for the development of incentive hope, which can be shown for different survival-related parameters (especially CS reliability, short delays, and additional information). Incentive hope is believed to increase food-seeking behavior and, when food is in sufficient amounts in the environment, food consumption as well. As a result, the animals seeking food items whose uncertainty is unavoidable have the opportunity to increase their fat reserves and/or to hoard more items. Autoshaping consists of a special case, in which incentive hope is only produced for CS reliability, and the experimental procedure does not allow the animal to increase reward rate. CS **= conditioned stimulus.**



<CAPT>Fig. 5. Portion of an environment traveled depending on its density of food. (A) Safe environment. (B) Moderately unpredictable environment. (C) Highly unpredictable environment. The safe environment was less explored than the two unpredictable environments. In each environment, the colored squares represent the total number of squares crossed by the eight foragers (a square was colored when at least one of the eight foragers entered, and consisted of a surface of 50 steps × 50 steps).



<CAPT>Fig. 6. The beneficial effects of increased food seeking imply that food amounts remain within a biologically acceptable range. (A) Compared with foragers exposed to a safe environment (group S-800), more fat is stored over time in group U-200, but there is a gradual loss of initial fat reserves in group U-60. (B) When food is not available, the ability to survive for longer periods = is proportional to the amount of fat stored. (C) Overall group comparisons indicate that fat reserves were higher in U-200 and lower in U-60 compared with S-800 foragers. (D) The number of food items consumed was higher in U-200 and lower in U-60 compared with S-800 foragers. (E) Motivational strength in seeking food was higher in U-200 than in S-800 foragers, but it even reached a greater intensity in U-60 foragers. (F) Mass-dependent predation risk was higher in U-200 foragers (because they gained weight) and lower in U-60 (because they lost weight) compared with S-800 foragers. Each data point on the abscissa must be multiplied by 50 to obtain the number of steps actually traveled (total: 60 × 50 = 3,000 steps).