Joanne Kwon

PID #: A15359545

**Part I**

**QUESTION #1**

Code:

i = 0;

while i <= 20

display (i\*5)

i = i+1;

end

**QUESTION #2**

* Removing the line ‘i=i+5’ outputs an infinite amount of ‘i=0’. This is because if ‘i=i+5’ is removed ‘i’ is always 0. This means i is always less than 100, so the while loop continues infinitely.

**QUESTION #3**

Code:

for i=0:100

display (i\*5);

end

**QUESTION #4**

* When testing the previous code I found that the values differ from the original while loop because the values range from 0 to 500 instead of 0 to 100. This is because the for loop asks for 100 values starting from 0 and increasing by 5.

**QUESTION #5**

Code:

vec = [0 1 3 6 7];

i = 1;

while i <= size(vec, 2)

vec(i) = vec(i) +2;

end

**QUESTION #6**

* If we reverse the order of the two lines in the body of the loop there is an error because the index exceeds the matrix dimensions.

**QUESTION #7**

Code:

vec = [0 1 3 6 7];

newvec = [ ];

for i = 1:size(vec,2)

newvec(i) = vec(i);

i = i + 1;

end

**QUESTION #8**

Code:

image = imread ('filename.jpg');

i = 1;

while i >= floor(size(image, 1)/2)

image(size(image, 1)-i+1, :, :) = image (i, :, :);

i = i + 1;

end

imshow(image);

**Part 2**

**QUESTION #9**

* A for loop is most likely the best option for the function ‘IsInSnake’ because the function needs to loop over a certain amount of times (the length of the snake). A while loop is most likely the best option for the function ‘GetFood’ because the function needs to loop to make sure that the the food generates outside of the snake. A switch statement is most likely the best option for the function ‘MoveHead’ because the function needs to make the snake move based on a certain movement (left, right, up, and down).

**QUESTION #10**

snake = [2,7;2,6;2,5;2,4;3,4;4,4;4,3];

IsInSnake (snake, [2,6]); → 1

IsInSnake (snake, [4,4]); → 1

IsInSnake (snake, [2,3]); → 0

IsInSnake (snake, [4,2]); → 0

GetFood (snake);

* Possible output → [2,3] and [4,2]
* Not possible output → [2,6] and [4,4]

MoveHead ([2,3], ‘up’); → [1,3]

MoveHead ([3,4], ‘down’); → [4,4]

MoveHead ([2,4], ‘left’); → [2,3]

MoveHead ([2,5], ‘right’); → [2,6]

**IsInSnake**

function [trueORfalse] = IsInSnake(snake, p)

% Name: Joanne Kwon

% The function 'IsInSnake' returns true or false depending on whether or

% not a specific row/column value occurs somewhere in the snake.

% This function takes in the arguments snake and p. The argument snake is

% an Lx2 matrix that stores the positions of the snake. The argument p is

% a query point (vector of length 2) that represents a position to be

% checked. This function should return true if the row and column values

% in p match those of any snake segment and return false otherwise.

trueORfalse = 0;

for i = 1:size(snake, 1)

if (snake(i,1) == p(1) && snake(i,2) == p(2))

trueORfalse = 1;

end

end

end

**GetFood**

function [p] = GetFood(snake)

% Name: Joanne Kwon

% The function 'GetFood' generates and returns a point (vector of length 2)

% with the condition that it does not overlap with any segment from snake.

% This function takes the argument snake, which is an Lx2 matrix that

% stores the position of the snake. The function generates new positions

% at a random point and generates a point with integers between 1 and

% 100. The function also checks if this point overlaps with snake and

% generates a new point if it does.

p = randi([1,100],1,2);

while IsInSnake(snake, p) == 1

p = randi([1,100],1,2);

end

end

**MoveHead**

function [NewPoint] = MoveHead(p, direction)

% Name: Joanne Kwon

% The function 'MoveHead' moves the snake based on the position of the head

% of the snake and the direction it's moving.

% This function takes in the arguments p and direction. The argument p is

% a point (vector of length 2) denoting the position of the head of the

% snake. The argument direction is a string that takes on the values

% 'up', 'down', 'left', or 'right' denoting the direction the snake is

% currently moving.

switch direction

case 'up'

NewPoint = [p(1)-1, p(2)];

case 'down'

NewPoint = [p(1)+1, p(2)];

case 'left'

NewPoint = [p(1), p(2)-1];

case 'right'

NewPoint = [p(1), p(2)+1];

end

end