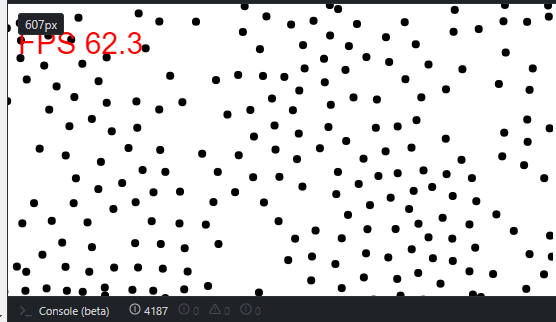
1. Avoidance Force-based Crowd Simulation (distance focused)  
   Implement the avoidance force-based crowd simulation method.The method is based on the distance between the agents in the current time step. An easy path forward would be to convert the linked Javascript code. the agents should roam around in space, and if they reach one of the edges, they would be transported to the other side as in a “round earth” scenario. Note that the linked Javascript code is not perfect, and might contain bugs. *Can you think of any way to improve it?* Test your code with at least 300 agents.

|  |
| --- |
| We recommend your animation starts by running with the `-favoid` argument  >python pj -favoid  You may also create a stand alone file for this task: >python pj\_part2.py Note that alternatively you might implement your code with in C/C++/Java |



var d = document,

canvas = d.body.appendChild( d.createElement( 'canvas' ) ),

ctx = canvas.getContext( '2d' ),

time = 0,

w = canvas.width = innerWidth,

h = canvas.height = innerHeight,

m = Math,

cos = m.cos,

sin = m.sin,

PI = m.PI,

lastTime = (new Date()).getTime(),

frames = 0,

totalTime = 0,

updateTime = 0,

updateFrames =0;

const times = [];

var fps = 0.0;

var fps\_avg = 0.0;

function refreshLoop() {

var now = (new Date()).getTime();

delta = now-lastTime;

lastTime = now;

totalTime+=delta;

frames++;

updateTime+=delta;

updateFrames++;

if(updateTime > 1000) {

fps = 1000\*frames/totalTime;

fps\_avg = 1000\*updateFrames/updateTime;

updateTime = 0;

updateFrames =0;

}

}

var numAgents = 2\* Math.floor(w/5);

var agents = [];

const RADIUS = 4;

const AGENTSIZE = RADIUS \* 2;

const d\_h = AGENTSIZE \* 3;

const TIME\_STEP = 0.5;

const MAX\_SPEED = 1.5;

const MAX\_FORCE = 0.01;

function distance(x1, y1, x2, y2) {

return Math.sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1));

}

function agentAtPoint(x, y) {

for (var i=0; i<numAgents; i++) {

if (distance(x, y, agents[i].x, agents[i].y) < 10) {

return true;

}

}

return true;

}

function makeAgent() {

var vx = (Math.random()\*2+0.5)\*(Math.random() < 0.5 ? 0.1 : -0.1);

var vy = (Math.random()\*2+0.5)\*(Math.random() < 0.5 ? 0.1 : -0.1);

return {

x: Math.random()\*w,

y: Math.random()\*h,

vx: vx,

vy: vy,

vx\_goal:(Math.random()\*2+0.5)\*(Math.random() < 0.5 ? 0.1 : -0.1),

vy\_goal: (Math.random()\*2+0.5)\*(Math.random() < 0.5 ? 0.1 : -0.1),

aggro: Math.random() > 0.5 ? true : false,

draw: function() {

ctx.beginPath();

ctx.arc(this.x, this.y, RADIUS, 0, 2\*PI, false);

if (this.aggro) {

ctx.fillStyle = 'black';

}

else {

ctx.fillStyle = 'black';

}

ctx.fill();

},

step: function(i ) {

/\*moves agents to the other side of the canvas\*/

if(this.x < -AGENTSIZE)

{

this.x = w;

}

else if(this.x > w+AGENTSIZE)

{

this.x = 0;

}

if(this.y < -AGENTSIZE)

{

this.y = h;

}

else if(this.y > h+AGENTSIZE)

{

this.y = 0;

}

/\* \*/

var v\_x = this.vx;

var v\_y = this.vy;

zeta = 1.0023;

var f\_goal\_x = (this.vx\_goal - v\_x) / zeta;

var f\_goal\_y = (this.vy\_goal - v\_y) / zeta;

if(this.vx\_goal > v\_x)

{

console.log("alert" );

}

var f\_avoid\_x = 0;

var f\_avoid\_y = 0;

var f\_avoid\_ctr = 0;

var interacting\_agents\_cntr = 0;

for(var j=0; j<numAgents; j++)

{

if(i === j ) { continue; }

var dist = distance(agents[i].x, agents[i].y, agents[j].x, agents[j].y);

if(dist > 0 && dist < d\_h)

{

var d\_ab = Math.max(dist - AGENTSIZE,0.001);

// why did I choose 0.0001?

var k = Math.max(d\_h - d\_ab, 0);

var x\_ab = (agents[i].x - agents[j].x)/dist;

var y\_ab = (agents[i].y - agents[j].y)/dist;

interacting\_agents\_cntr +=1;

f\_avoid\_x += k \* x\_ab / d\_ab;

f\_avoid\_y += k \* y\_ab / d\_ab;

f\_avoid\_ctr=f\_avoid\_ctr+1;

}

}

if(f\_avoid\_ctr > 0 )

{

f\_avoid\_x = f\_avoid\_x / f\_avoid\_ctr;

f\_avoid\_y = f\_avoid\_y / f\_avoid\_ctr;

}

//f\_goal\_x =0 ; f\_goal\_y = 0;

var force\_sum\_x = f\_goal\_x + f\_avoid\_x;

var force\_sum\_y = f\_goal\_y + f\_avoid\_y;

var f\_avoid\_mag = Math.sqrt(force\_sum\_x\*force\_sum\_x + force\_sum\_y\*force\_sum\_y);

if(f\_avoid\_mag > MAX\_FORCE )

{

force\_sum\_x = MAX\_FORCE\* force\_sum\_x/ f\_avoid\_mag ;

force\_sum\_y = MAX\_FORCE\* force\_sum\_y / f\_avoid\_mag ;

}

v\_x += TIME\_STEP \* force\_sum\_x;

v\_y += TIME\_STEP \* force\_sum\_y;

var speed = Math.sqrt(v\_x\*v\_x + v\_y\*v\_y);

if(speed > MAX\_SPEED)

{

v\_x = MAX\_SPEED \* v\_x / speed ;

v\_y = MAX\_SPEED \* v\_y / speed;

}

this.vx = v\_x;

this.vy = v\_y;

this.x += TIME\_STEP\* v\_x;

this.y += TIME\_STEP\* v\_y;

}

}

}

// make N agents

for (var i=0; i<numAgents; i++) {

agents.push(makeAgent());

}

//fps counter

// The main animation loop

setInterval( function() {

// Clear

canvas.width = canvas.width;

time += TIME\_STEP;

for (var i=0; i<numAgents; i++) {

agents[i].step(i);

agents[i].draw();

}

refreshLoop();

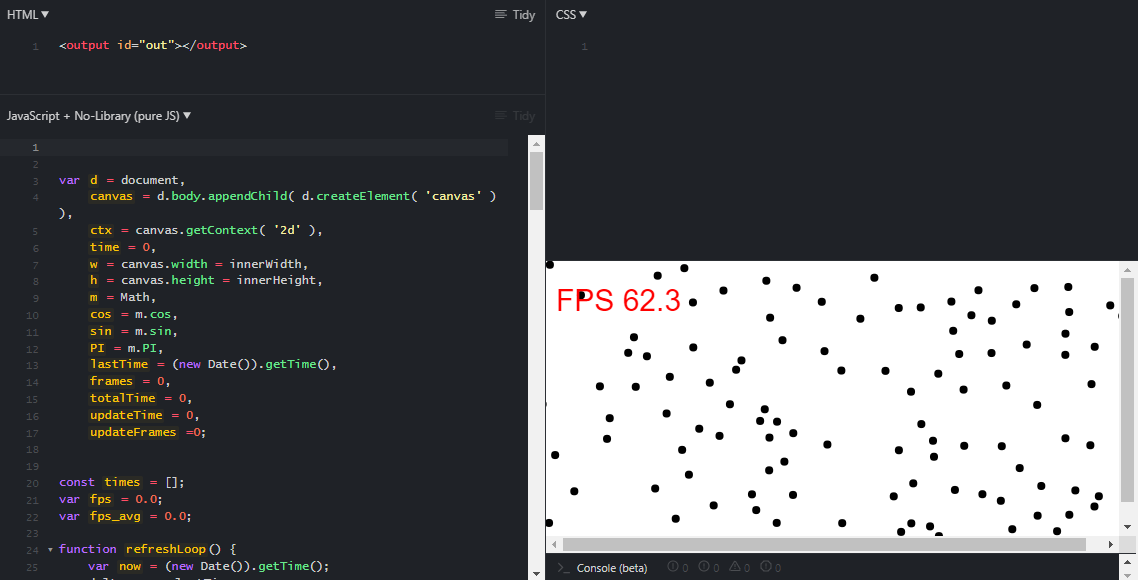
ctx.font = "30px Arial";

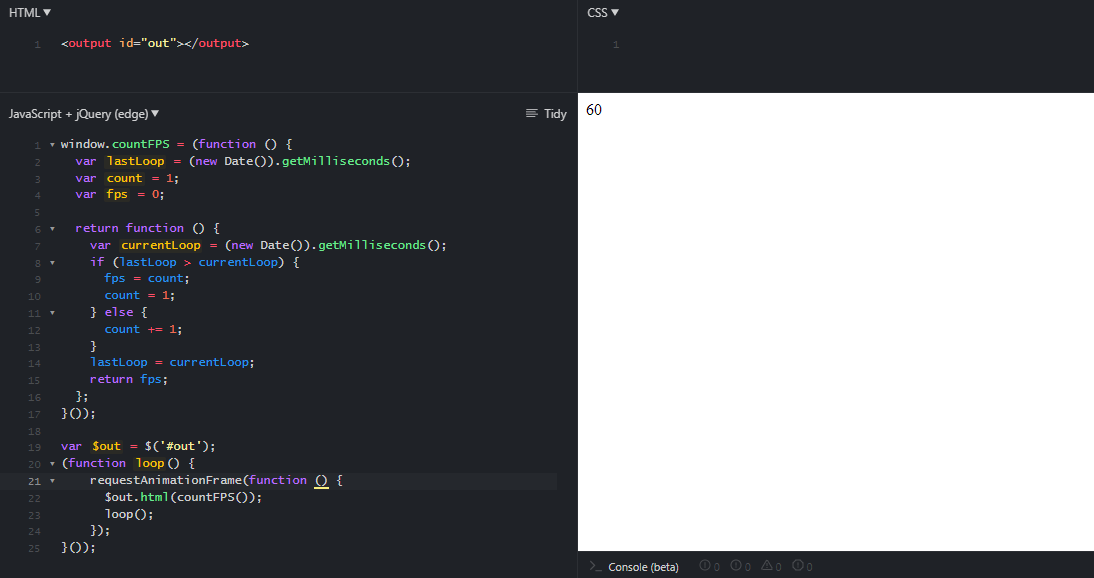
ctx.fillStyle = "red";

ctx.fillText("FPS " + fps.toFixed(1), 10, 50);

}, 16 )

1. Calculating Frames Per Second     
   \* With the crowd simulation you implemented in step 1, calculate the number of frames per second. Try to display the frames per second either in the console or on the screen and not flicker, like here:





var d = document,

canvas = d.body.appendChild( d.createElement( 'canvas' ) ),

ctx = canvas.getContext( '2d' ),

time = 0,

w = canvas.width = innerWidth,

h = canvas.height = innerHeight,

m = Math,

cos = m.cos,

sin = m.sin,

PI = m.PI,

lastTime = (new Date()).getTime(),

frames = 0,

totalTime = 0,

updateTime = 0,

updateFrames =0;

const times = [];

var fps = 0.0;

var fps\_avg = 0.0;

function refreshLoop() {

var now = (new Date()).getTime();

delta = now-lastTime;

lastTime = now;

totalTime+=delta;

frames++;

updateTime+=delta;

updateFrames++;

if(updateTime > 1000) {

fps = 1000\*frames/totalTime;

fps\_avg = 1000\*updateFrames/updateTime;

updateTime = 0;

updateFrames =0;

}

}

function refreshLoop2() {

window.requestAnimationFrame(() => {

const now = performance.now();

while (times.length > 0 && times[0] <= now - 1000) {

times.shift();

}

times.push(now);

fps = times.length;

refreshLoop();

});

}

var numAgents = 1\* Math.floor(w/5);

var agents = [];

const RADIUS = 4;

const AGENTSIZE = RADIUS \* 2;

const d\_h = AGENTSIZE \* 5;

const TIME\_STEP = .1;

function distance(x1, y1, x2, y2) {

return Math.sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1));

}

function agentAtPoint(x, y) {

for (var i=0; i<numAgents; i++) {

if (distance(x, y, agents[i].x, agents[i].y) < 10) {

return true;

}

}

return true;

}

function makeAgent() {

var vx = (Math.random()\*2+0.5)\*(Math.random() < 0.5 ? 0.1 : -0.1);

var vy = (Math.random()\*2+0.5)\*(Math.random() < 0.5 ? 0.1 : -0.1);

return {

x: Math.random()\*w,

y: Math.random()\*h,

vx: vx,

vy: vy,

vx\_: vx,

vy\_: vy,

aggro: Math.random() > 0.5 ? true : false,

draw: function() {

ctx.beginPath();

ctx.arc(this.x, this.y, RADIUS, 0, 2\*PI, false);

if (this.aggro) {

ctx.fillStyle = 'black';

}

else {

ctx.fillStyle = 'black';

}

ctx.fill();

},

step: function(i ) {

/\*moves agents to the other side of the canvas\*/

if(this.x < -AGENTSIZE)

{

this.x = w;

}

else if(this.x > w+AGENTSIZE)

{

this.x = 0;

}

if(this.y < -AGENTSIZE)

{

this.y = h;

}

else if(this.y > h+AGENTSIZE)

{

this.y = 0;

}

/\* \*/

v\_x = this.vx;

var v\_y = this.vy;

var f\_avoid\_x = 0;

var f\_avoid\_y = 0;

var interacting\_agents\_cntr = 0;

for(var j=0; j<numAgents; j++)

{

if(i === j ) { continue; }

var dist = distance(agents[i].x, agents[i].y, agents[j].x, agents[j].y);

if(dist > 0 && dist < d\_h)

{

var d\_ab = Math.max(dist - AGENTSIZE,0.001);

// why did I choose 0.0001?

var k = Math.max(d\_h - d\_ab, 0);

var x\_ab = (agents[i].x - agents[j].x)/dist;

var y\_ab = (agents[i].y - agents[j].y)/dist;

interacting\_agents\_cntr +=1;

f\_avoid\_x += k \* x\_ab / d\_ab;

f\_avoid\_y += k \* y\_ab / d\_ab;

}

}

var f\_avoid\_mag = Math.sqrt(f\_avoid\_x\*f\_avoid\_x + f\_avoid\_y\*f\_avoid\_y);

if(f\_avoid\_mag > 0.01)

{

f\_avoid\_x /=f\_avoid\_mag;

f\_avoid\_y /=f\_avoid\_mag;

}

v\_x += TIME\_STEP \* f\_avoid\_x;

v\_y += TIME\_STEP \* f\_avoid\_y;

this.x += v\_x;

this.y += v\_y;

}

}

}

// make N agents

for (var i=0; i<numAgents; i++) {

agents.push(makeAgent());

}

//fps counter

// The main animation loop

setInterval( function() {

// Clear

canvas.width = canvas.width;

time += TIME\_STEP;

for (var i=0; i<numAgents; i++) {

agents[i].step(i);

agents[i].draw();

}

refreshLoop();

ctx.font = "30px Arial";

ctx.fillStyle = "red";

ctx.fillText("FPS " + fps.toFixed(1), 10, 50);

}, 16 )

|  |
| --- |
| Your animation should start running interactively with the `-fps` argument  >python pj -fps  You may also create a stand alone file for this task: >python pj\_part2.py Note that alternatively you might implement your code with in C/C++/Java |

1. Spatial Hash for Collision Detection  
   Implement spatial hash for making interagent collision detection more efficient. Test your implementation by checking the number of frames per second, which should be a higher number than the non-spatial hash version of your code. This will be true if you have many agents roaming around. The following links would be helpful:  
   \* Spatial Partitioning from Real Time Collision Detection book https://drive.google.com/file/d/1maTSl8WiQ6W3ivmGE5hiKBzCwFMwzuEs/view

\* Spatial Partition from Game Programming Patterns book [(2)](https://gameprogrammingpatterns.com/spatial-partition.html) ]--<https://gameprogrammingpatterns.com/spatial-partition.html>  
\* Spatial Hashing for fast 2d collision detection blog [(3)](https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/)  --<https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/>

|  |
| --- |
| Your animation should start running interactively with the `shash` argument  >python pj -shash  You may also create a stand alone file for this task: >python pj\_part3.py Note that alternatively you might implement your code with in C/C++/Java |

**Source code and Implementation Instructions**

* Start early! It might take some time to setup your opengl pipeline to compile properly.
* Your solution must include source code in Python / C++ / Java / WebGL.
* OpenGL packages not listed below are not allowed.
* External packages not listed below are not allowed.
* Python Instructions:
  + **Provide source code, and a requirments.txt file (e.g.** [**here**](https://medium.com/@boscacci/why-and-how-to-make-a-requirements-txt-f329c685181e)**)** you used.
  + Note that you may need to install PyOpenGL\_accelerate
  + As usual,  remember to set up a python virtual environment and then pip install the packages above.
* Java Instructions:
  + **Provide source code, and compiled Jar file for each task**
  + You are allowed to use:
* LWJGL3
  + Download<https://github.com/LWJGL/lwjgl3/releases>
  + Follow<https://www.lwjgl.org/guide>
    - If you are on a Mac, might need to add [this](https://stackoverflow.com/questions/56511732/lwjgl-doesnt-work-on-mac-due-to-some-apple-libraries-objective-c-misbehaving) to your code if it doesn’t compile
* JOGL
  + See<https://www.tutorialspoint.com/jogl/jogl_quick_guide.htm>
* C\C++ Instructions:
  + **Provide your source code, compilation line/makefile. Your code should compile on Linux.**
* You are allowed to use:
  + GLUT  
    <https://www.programming-techniques.com/2011/12/glut-tutorial-2d-animation.html>
  + GLFW   
    <https://www.glfw.org/docs/latest/quick.html>