

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

EVOLUTIONARY COMPUTING

LABORATORY SESSION #07: PARTICLE SYSTEMS

STUDENT: VARGAS ROMERO ERICK EFRAIN

PROFESSOR: ROSAS TRIGUEROS JORGE LUIS

PRACTICE COMPLETION DATE: NOVEMBER 8, 2021

REPORT DELIVERY DATE: NOVEMBER 04, 2021

## Theoretical framework

### 1.1 Particle Systems

In 1982, William T. Reeves, a researcher at Lucasfilm Ltd., was working on the film *Star Trek II: The Wrath of Khan*. Much of the movie revolves around the Genesis Device, a torpedo that when shot at a barren, lifeless planet has the ability to reorganize matter and create a habitable world for colonization. During the sequence, a wall of fire ripples over the planet while it is being “terraformed.” The term particle system, an incredibly common and useful technique in computer graphics, was coined in the creation of this particular effect.

*“A particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system.”*

—William Reeves

Since the early 1980s, particle systems have been used in countless video games, animations, digital art pieces, and installations to model various irregular types of natural phenomena, such as fire, smoke, waterfalls, fog, grass, bubbles, and so on.

### 1.2 Basic Model of Particle Systems

A particle system is a collection of many particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system.

To compute each frame in a motion sequence, the following sequence of steps is performed:

1. New particles are generated into the system.
2. Each new particle is assigned its individual attributes.

3. Any particles that have existed within the system past their prescribed lifetime are extinguished.
4. The remaining particles are moved and transformed according to their dynamic attributes.
5. An image of the living particles is rendered in a frame buffer.

The particle system can be programmed to execute any set of instructions at each step. Because it is procedural, this approach can incorporate any computational model that describes the appearance of dynamics of the object. For example, the motions and transformations of particles could be tied to the solution of a system of partial differential equations or particle attributes could be assigned on basis of statistical mechanics. We can therefore, take advantage of models which have been developed in other scientific or engineering disciplines [1].

## Material and equipment

The necessary material for this practice is:

- A computer with the latest *Python* stable version installed
- A text editor

Or is possible to use the google site <https://colab.research.google.com/> that allows us to use a virtual machine with an *Ubuntu* operative system with *Python* installed.

## Practice development

### 3.1 Particle Systems

To develop this practice we used the Google platform called *Colab* as this platform uses a virtual machine with linux (specifically Ubuntu) we can install some packets and of course we can verify if we have already *Python* installed. To check it we must use the command:

```
1 python --version
```

If we run this command in our linux terminal using *Colab* we can see the next as the result:

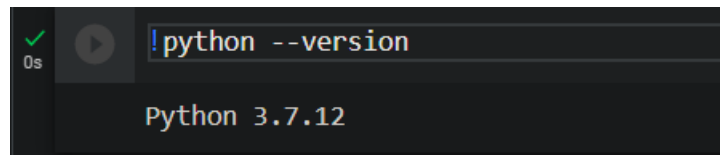


Figure 3.1: Verifying python version

### 3.2 Implement a particle system in 2D with nearest neighbour interactions in a grid

To solve this practice we based in the code provided by the professor. The mexican wave we already known that works in one dimension and works with the interaction between adjacent particles. So to solve this first task we have something very similar but instead of one dimension we have two dimensions. In code to make easier to understand the behaviour of the system we built a class called *Particle* this class contains a *constructor* that receives the initial position of the particle, radius (can be also the force of the particle) and a boolean state called *isGrowing* that indicates that the  $i$ -th

### 3.2. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH NEAREST NEIGHBOUR INTERACTIONS IN A GRID5

particle is changing or can be affected by their neighbours. This class also contains a method called *draw*, basically, this function draws a particle in the space.

```
1 class Particle:
2     #Constructor
3     def __init__(self, x, y):
4         self.radius = 0.0
5         self.x = x
6         self.y = y
7         self.isGrowing = True
8
9     def draw(self, img):
10        color = (255, 255, 255)
11        cv2.circle(img, (int(self.x), int(self.y)), int(self.radius), color, -1)
```

Other interesting function in this problem is the called *nextIteration*. This function makes possible to change every particle in the space, we iterate particle by particle and for each particle we get the "force" of the neighbors and we add that "force" in the current particle but of course the force is decreased in every iteration.

```
1 def nextIteration(particles):
2     global maxRadius, activeParticles, decayValue, increaseConstant
3
4     neighbors = [[1,0], [0,1], [-1,0], [0,-1]]
5     xParticles = len(particles)
6     yParticles = len(particles[0])
7     activeParticles = 0
8
9     for x in range(0, xParticles):
10        for y in range(0, yParticles):
11            if particles[x][y].isGrowing:
12                for xOffset, yOffset in neighbors:
13                    #We add to the current particle the "force" applied by the
14                    #neighbours.
15                    if (0 <= x + xOffset < xParticles) and (0 <= y + yOffset <
16                    yParticles):
17                        particles[x][y].radius += increaseConstant * particles[x +
18                        xOffset][y + yOffset].radius
19
20                    #We reach the maximum radius of the current particle we don't need to
21                    #use it again
22                    if particles[x][y].isGrowing and particles[x][y].radius >= maxRadius:
23                        particles[x][y].isGrowing = False
24
25                    #If the current particle is not growing we set the particle to the
26                    #initial state (talking about "force")
27                    if not particles[x][y].isGrowing:
28                        if particles[x][y].radius <= 1.0:
29                            particles[x][y].radius = 0
30
31                    #We count the number of particles that are in use
32                    if particles[x][y].radius > 1.0:
33                        activeParticles += 1
34
35                    #Decay
36                    particles[x][y].radius *= decayValue
```

The entire code to solve this task is here to get a better understanding about the functionality:

### 3.2. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH NEAREST NEIGHBOUR INTERACTIONS IN A GRID6

```
1 import cv2
2 import numpy as np
3 from IPython import display as display
4 import ipywidgets as ipw
5 import PIL
6 import random
7 import time
8 from io import BytesIO
9
10 class Particle:
11     #Constructor
12     def __init__(self, x, y):
13         self.radius = 0.0
14         self.x = x
15         self.y = y
16         self.isGrowing = True
17
18     def draw(self, img):
19         color = (255, 255, 255)
20         cv2.circle(img, (int(self.x), int(self.y)), int(self.radius), color, -1)
21
22 #Creates all the possible particles in the space
23 def generateParticles(x0, y0, maxX, maxY):
24
25     global spaceBetweenParticles
26
27     xValues = []
28     yValues = []
29     particles = []
30
31     xi = x0
32     while xi > spaceBetweenParticles:
33         xValues.append(xi)
34         xi -= spaceBetweenParticles
35     xi = x0
36     while xi < maxX:
37         xValues.append(xi)
38         xi += spaceBetweenParticles
39     yi = y0
40     while yi > spaceBetweenParticles:
41         yValues.append(yi)
42         yi -= spaceBetweenParticles
43     yi = y0
44     while yi < maxY:
45         yValues.append(yi)
46         yi += spaceBetweenParticles
47
48     xValues.sort()
49     yValues.sort()
50
51     xSize = len(xValues)
52     ySize = len(yValues)
53
54     for x in range(0, xSize):
55         currParticles = []
56         for y in range(0, ySize):
57             currParticles.append(Particle(xValues[x], yValues[y]))
58         particles.append(currParticles)
```

### 3.2. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH NEAREST NEIGHBOUR INTERACTIONS IN A GRID

```
59
60     return particles
61
62 def chooseRandomParticle(particles):
63     xRandomPosition = random.randrange(1, len(particles) - 1)
64     yRandomPosition = random.randrange(1, len(particles[0]) - 1)
65     particles[xRandomPosition][yRandomPosition].radius = 1
66
67 def nextIteration(particles):
68     global maxRadius, activeParticles, decayValue, increaseConstant
69
70     neighbors = [[1,0], [0,1], [-1,0], [0,-1]]
71     xParticles = len(particles)
72     yParticles = len(particles[0])
73     activeParticles = 0
74
75     for x in range(0, xParticles):
76         for y in range(0, yParticles):
77             if particles[x][y].isGrowing:
78                 for xOffset, yOffset in neighbors:
79                     #We add to the current particle the "force" aplyied by the
neighbours.
80                     if(0 <= x + xOffset < xParticles) and (0 <= y + yOffset <
yParticles):
81                         particles[x][y].radius += increaseConstant * particles[x +
xOffset][y + yOffset].radius
82
83                     #We reach the maximum radius of the current particle we don't need to
use it again
84                     if particles[x][y].isGrowing and particles[x][y].radius >= maxRadius:
85                         particles[x][y].isGrowing = False
86
87                     #If the current particle is not growing we set the particle to the
initial state (talking about "force")
88                     if not particles[x][y].isGrowing:
89                         if particles[x][y].radius <= 1.0:
90                             particles[x][y].radius = 0
91
92                     #We count the number of particles that are in use
93                     if particles[x][y].radius > 1.0:
94                         activeParticles += 1
95
96                     #Decay
97                     particles[x][y].radius *= decayValue
98
99 def drawParticles(particles, img):
100
101     xParticles = len(particles)
102     yParticles = len(particles[0])
103
104     for x in range(0, xParticles):
105         for y in range(0, yParticles):
106             particles[x][y].draw(img)
107
108 #Or rows
109 height = 500
110 #Or cols
111 width = 600
```



### 3.2. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH NEAREST NEIGHBOUR INTERACTIONS IN A GRID8

```
112 #Max particle radius
113 maxRadius = 10
114 #Max particle radius
115 spaceBetweenParticles = maxRadius * 2 + 1
116 activeParticles = 1
117 decayValue = 0.9
118 increaseConstant = 0.15
119
120 x0 = width // 2
121 y0 = (height) // 2
122
123 maxIterations = 200
124
125 wIm = ipw.Image()
126 display.display(wIm)
127
128 #We create our grid
129 image = np.zeros((height + spaceBetweenParticles, width + spaceBetweenParticles, 3),
130                 dtype = "uint8")
131 #Creating all the particles
132 #We add an offset in height because without that we can't see the legend (Iteration:
133 n)
134 particles = generateParticles(x0, y0, width, height - (30 + maxRadius))
135
136 xParticles = len(particles)
137 yParticles = len(particles[0])
138
139 drawParticles(particles, image)
140
141 for iteration in range(0, 200):
142     image[:] = (0, 0, 0)
143
144     if activeParticles == 0:
145         for x in range(0, xParticles):
146             for y in range(0, yParticles):
147                 particles[x][y].isGrowing = True
148
149     chooseRandomParticle(particles)
150
151     nextIteration(particles)
152     drawParticles(particles, image)
153     #Show the current iteration in the image
154     cv2.putText(image, "Iteration: " + str(iteration + 1), (spaceBetweenParticles,
155 height - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))
156     #Show the current iteration in the image
157     cv2.putText(image, "Particles: " + str(activeParticles), (250 +
158 spaceBetweenParticles, height - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255,
159 255))
160
161     pilIm = PIL.Image.fromarray(image, mode = "RGB")
162
163     with BytesIO() as fOut:
164         pilIm.save(fOut, format = "png")
165         byPng = fOut.getvalue()
166
167     # set the png bytes as the image value;
168     # this updates the image in the browser.
169     wIm.value = byPng
```

```
165 time.sleep(0.15)
```

### 3.3 Implement a Particle System in 2D with collisions (A box with a small window)

To solve this task we used the code provider by the teacher, This task was a little bit more complicated than the previous one because of the physics implied in the behaviour of each particle. Talking about the physics of this task, we used the first and second Newton's law, and as we are using a 2D particle system is mandatory to use vectorial calculus to modelate all the physical phenomena

Something really important about this task is to make all the particles move to the right, because in the right side of the box, we created a small exit in order to see the behaviour of the particles. So to solve this we must increase the force in the  $y$  component of the vector and decrease this force in each iteration this step is done after calculate the force applied by each particle around the current particle.

```
1 for count in range(MaxIterations):
2     img[:] = (0, 0, 0)
3     for i in range(NumParticles):
4         for j in range(NumParticles):
5             if i != j:
6                 Fij = particles[i].calculateForce(particles[j])
7                 particles[i].force += Fij
8                 #As we already known we must guide all the particles to the right
9                 result = exitAtractor - particles[i].position
10                magnitudeResult = np.linalg.norm(result)
11                if magnitudeResult <= 0:
12                    magnitudeResult = 1
13                #Calculating the force of the vector
14                resultForce = result / magnitudeResult
15                #DECAY
16                resultForce[0] *= 0.001
17                resultForce[1] *= 0.009
18                particles[i].force += resultForce
19
20 for particle in particles:
21     particle.updateSpeed()
22     particle.updatePosition()
23     particle.draw(x0, y0, img)
24     particle.force[:] = 0
```

The rest of the code is almost the same, here we add the entire code in order to check carefully how this task works.

```
1 import cv2
2 import numpy as np
3 from IPython import display as display
4 import ipywidgets as ipw
5 import PIL
6 from io import BytesIO
7 import math
8 import random
9 import time
```

### 3.3. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW)10

```
10
11 class Particle:
12     MaxV = np.sqrt(2)
13
14     def __init__(self, x, y, speedX, speedY, radius, WallParticle=False):
15         self.position = np.array([float(x), float(y)])
16         self.speed = np.array([float(speedX), float(speedY)])
17         self.WallParticle = WallParticle
18         self.force = np.array([0.0, 0.0])
19         self.radius = radius
20
21     #By definition an unit vector it's magnitude must be 1 otherwise it isn't a unit
    vector
22     def normalizeVector(self, x):
23         norm = np.linalg.norm(x)
24         if norm == 0.0:
25             return x * np.inf
26         return x / norm
27
28     def calculateForce(self, r2):
29         if self.WallParticle == True:
30             return np.array([0.0, 0.0])
31         result = self.position - r2.position
32         r12magnitude = np.linalg.norm(result)
33         if r12magnitude <= r2.radius + self.radius:
34             #We get the unitary vector and then we split by the magnitude of R where
    R is a new vector from r1 to r2
35             return self.normalizeVector(result) / (r12magnitude ** 2) *
    wallParticlesSize * 20
36         return np.array([0.0, 0.0])
37
38     def updatePosition(self):
39         if self.WallParticle == True:
40             return
41         self.position += self.speed
42         return
43
44     def updateSpeed(self):
45         if self.WallParticle == True:
46             return
47         self.speed += self.force
48         vmag = np.linalg.norm(self.speed)
49
50         if vmag > self.MaxV:
51             self.speed = self.normalizeVector(self.speed) * self.MaxV
52         return
53
54     def draw(self, x0, y0, img):
55         if self.WallParticle == True:
56             color = (0, 255, 255)
57         else:
58             color = (255, 255, 255)
59         cv2.circle(img, (int(x0 + self.position[0]), int(y0 - self.position[1])),
    self.radius, color, -1)
60         return
61
62
63 particles = []
```

### 3.3. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW)11

```
64 def lineOfWallParticles(x1, y1, x2, y2, N):
65     global particles
66     x = np.linspace(x1, x2, N)
67     y = np.linspace(y1, y2, N)
68     for i in range(N):
69         particles.append(Particle(x[i], y[i], 0, 0, wallParticlesSize, WallParticle=
        True))
70
71
72 wIm = ipw.Image()
73 display.display(wIm)
74
75 maxX = 500
76 maxY = 500
77 x0 = int(maxX / 2)
78 y0 = int(maxY / 2)
79 nParticles = 70
80 particleSize = 7
81 wallParticlesSize = 15
82
83 img = np.zeros((500, 500, 3), dtype="uint8")
84
85 #Upper wall
86 lineOfWallParticles(-150, 100, 150, 100, 20)
87 #Bottom wall
88 lineOfWallParticles(-150, -100, 150, -100, 20)
89 #Left wall
90 lineOfWallParticles(-150, 100, -150, -100, 20)
91 #Right wall
92 lineOfWallParticles(150, -100, 150, -30, 10)
93 lineOfWallParticles(150, 100, 150, 30, 10)
94 #Exit atractor
95 exitAtractor = np.array([300.0, 0.0])
96
97 for i in range(0, nParticles):
98     x = random.randrange(-150 + 2 * wallParticlesSize, -100 + 2 * wallParticlesSize)
99     y = random.randrange(-100 + 2 * wallParticlesSize, 100 - 2 * wallParticlesSize)
100     speedX = random.random()
101     speedY = random.random()
102     newParticle = Particle(x, y, speedX, speedY, particleSize)
103     particles.append(newParticle)
104
105
106 MaxIterations = 500
107 NumParticles = len(particles)
108
109 for count in range(MaxIterations):
110     img[:] = (0, 0, 0)
111     for i in range(NumParticles):
112         for j in range(NumParticles):
113             if i != j:
114                 Fij = particles[i].calculateForce(particles[j])
115                 particles[i].force += Fij
116                 #As we already known we must guide all the particles to the right
117                 result = exitAtractor - particles[i].position
118                 magnitudeResult = np.linalg.norm(result)
119                 if magnitudeResult <= 0:
120                     magnitudeResult = 1
```

### 3.4. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW AND AN OBSTACLE)

```
121         #Calculating the force of the vector
122         resultForce = result / magnitudeResult
123         #DECAY
124         resultForce[0] *= 0.001
125         resultForce[1] *= 0.009
126         particles[i].force += resultForce
127
128     for particle in particles:
129         particle.updateSpeed()
130         particle.updatePosition()
131         particle.draw(x0, y0, img)
132         particle.force[:] = 0
133
134     cv2.putText(img, "Iteration: " + str(count + 1), (20, 40), cv2.
135 FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))
136
137     pilIm = PIL.Image.fromarray(img, mode="RGB")
138     with BytesIO() as fOut:
139         pilIm.save(fOut, format="png")
140         byPng = fOut.getvalue()
141
142     # set the png bytes as the image value;
143     # this updates the image in the browser.
144     wim.value = byPng
```

## 3.4 Implement a Particle System in 2D with collisions (A box with a small window and an obstacle)

To this final task we used the same code than in the previous task, but with an extra particle that is static and works as an obstacle to the rest of particles that can move. As you can see below we only add an extra line with a particle.

```
1 #Obstacle
2 particles.append(Particle(100, 0, 0, 0, wallParticlesSize, WallParticle=True))
```

Here is the entire code to a better understanding

```
1 import cv2
2 import numpy as np
3 from IPython import display as display
4 import ipywidgets as ipw
5 import PIL
6 from io import BytesIO
7 import math
8 import random
9 import time
10
11 class Particle:
12     MaxV = np.sqrt(2)
13
14     def __init__(self, x, y, speedX, speedY, radius, WallParticle=False):
15         self.position = np.array([float(x), float(y)])
16         self.speed = np.array([float(speedX), float(speedY)])
17         self.WallParticle = WallParticle
18         self.force = np.array([0.0, 0.0])
```

### 3.4. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW AND AN OBSTACLE)

```
19     self.radius = radius
20
21     def normalizeVector(self, x):
22         norm = np.linalg.norm(x)
23         if norm == 0:
24             return x * np.inf
25         return x / norm
26
27     def calculateForce(self, r2):
28         if self.WallParticle == True:
29             return np.array([0.0, 0.0])
30         result = self.position - r2.position
31         r12magnitude = np.linalg.norm(result)
32         if r12magnitude <= r2.radius + self.radius:
33             #We get the unitary vector and then we split by the magnitude of R where
34             #R is a new vector from r1 to r2
35             return self.normalizeVector(result) / (r12magnitude ** 2) *
36             wallParticlesSize * 20
37         return np.array([0.0, 0.0])
38
39     def updatePosition(self):
40         if self.WallParticle == True:
41             return
42         self.position += self.speed
43         return
44
45     def updateSpeed(self):
46         if self.WallParticle == True:
47             return
48         self.speed += self.force
49
50         vmag = np.linalg.norm(self.speed)
51
52         if vmag > self.MaxV:
53             self.speed = self.normalizeVector(self.speed) * self.MaxV
54         return
55
56     def draw(self, x0, y0, img):
57         if self.WallParticle == True:
58             color = (0, 255, 255)
59         else:
60             color = (255, 255, 255)
61
62         cv2.circle(img, (int(x0 + self.position[0]), int(y0 - self.position[1])),
63                     self.radius, color, -1)
64         return
65
66 particles = []
67 def lineOfWallParticles(x1, y1, x2, y2, N):
68     global particles
69     x = np.linspace(x1, x2, N)
70     y = np.linspace(y1, y2, N)
71     for i in range(N):
72         particles.append(Particle(x[i], y[i], 0, 0, wallParticlesSize, WallParticle=
```

### 3.4. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW AND AN OBSTACLE)

```
73 wIm = ipw.Image()
74 display.display(wIm)
75
76 maxX = 500
77 maxY = 500
78 x0 = int(maxX / 2)
79 y0 = int(maxY / 2)
80 nParticles = 50
81 particleSize = 7
82 wallParticlesSize = 15
83
84 img = np.zeros((500, 500, 3), dtype="uint8")
85
86 #Upper wall
87 lineOfWallParticles(-150, 100, 150, 100, 20)
88 #Bottom wall
89 lineOfWallParticles(-150, -100, 150, -100, 20)
90 #Left wall
91 lineOfWallParticles(-150, 100, -150, -100, 20)
92 #Right wall
93 lineOfWallParticles(150, -100, 150, -30, 10)
94 lineOfWallParticles(150, 100, 150, 30, 10)
95 #Exit atractor
96 exitAtractor = np.array([300.0, 0.0])
97 #Obstacle
98 particles.append(Particle(100, 0, 0, 0, wallParticlesSize, WallParticle=True))
99
100 for i in range(0, nParticles):
101     x = random.randrange(-150 + 2 * wallParticlesSize, -100 + 2 * wallParticlesSize)
102     y = random.randrange(-100 + 2 * wallParticlesSize, 100 - 2 * wallParticlesSize)
103     speedX = random.random()
104     speedY = random.random()
105     newParticle = Particle(x, y, speedX, speedY, particleSize)
106     particles.append(newParticle)
107
108
109 MaxIterations = 500
110 NumParticles = len(particles)
111
112 for count in range(MaxIterations):
113     img[:] = (0, 0, 0)
114     for i in range(NumParticles):
115         for j in range(NumParticles):
116             if i != j:
117                 Fij = particles[i].calculateForce(particles[j])
118                 particles[i].force += Fij
119                 #As we already known we must guide all the particles to the right
120                 result = exitAtractor - particles[i].position
121                 magnitudeResult = np.linalg.norm(result)
122                 if magnitudeResult <= 0:
123                     magnitudeResult = 1
124                 #Calculating the force of the vector
125                 resultForce = result / magnitudeResult
126                 resultForce[0] *= 0.001
127                 resultForce[1] *= 0.009
128                 particles[i].force += resultForce
129
130     for particle in particles:
```

### 3.4. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW AND AN OBST

```
131     particle.updateSpeed()
132     particle.updatePosition()
133     particle.draw(x0, y0, img)
134     particle.force[:] = 0
135
136     cv2.putText(img, "Iteration: " + str(count + 1), (20, 40), cv2.
FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255))
137
138     pilIm = PIL.Image.fromarray(img, mode="RGB")
139     with BytesIO() as fOut:
140         pilIm.save(fOut, format="png")
141         byPng = fOut.getvalue()
142
143     # set the png bytes as the image value;
144     # this updates the image in the browser.
145     wIm.value = byPng
```



### 4.1 Implement a particle system in 2D with nearest neighbour interactions in a grid

As you can see in the img. 4.1 when we run the simulation randomly a particle is chosen and from that particle the neighbors are affected.

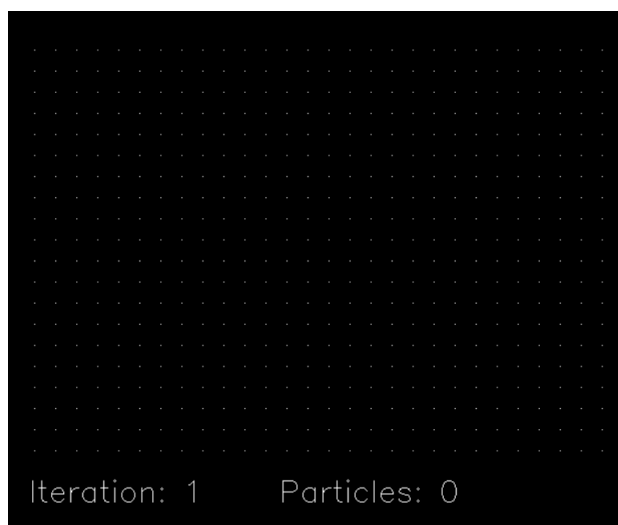


Figure 4.1: 2D Particle system with neighbour interactions, iteration 1

In the img 4.2 as you can see the program choose randomly a particle and now this particle affected

#### 4.1. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH NEAREST NEIGHBOUR INTERACTIONS IN A GRID17

more particles around of it and that particles affected other ones and so on, as you can see it the pattern grow.

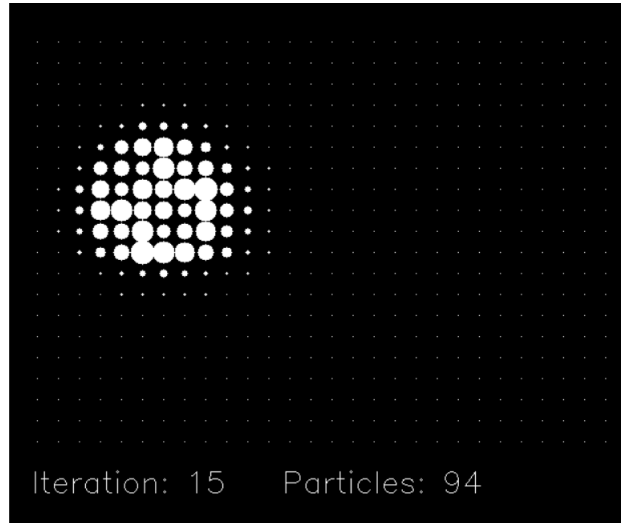


Figure 4.2: 2D Particle system with neighbour interactions, iteration 15

Now in img. 4.3 the pattern grow more but the first particles affected now are decaying.

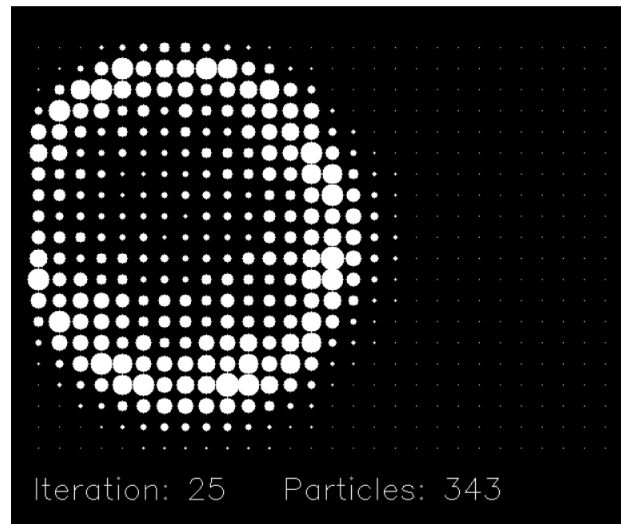


Figure 4.3: 2D Particle system with neighbour interactions, iteration 25

#### 4.1. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH NEAREST NEIGHBOUR INTERACTIONS IN A GRID18

In img. 4.4 and img. 4.5 we have the same behaviour than in the previous image. It looks like a wave.

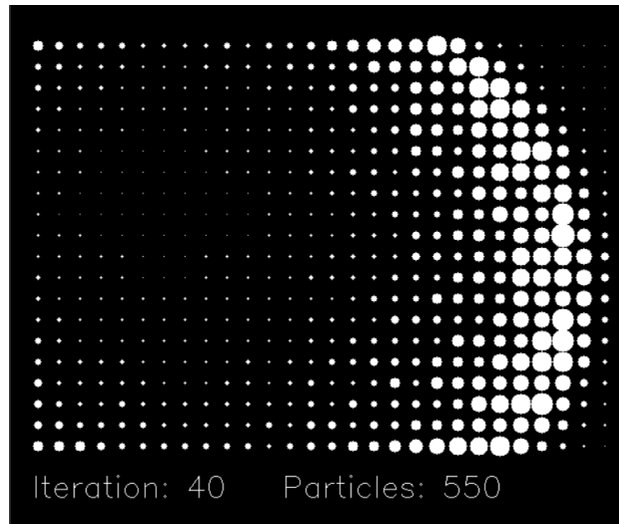


Figure 4.4: 2D Particle system with neighbour interactions, iteration 40

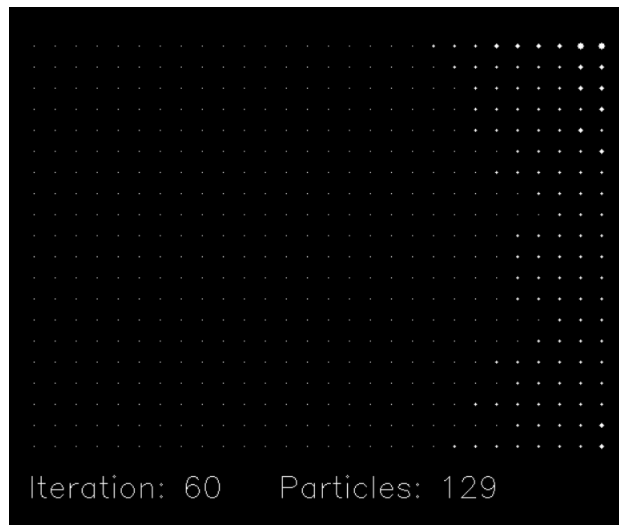


Figure 4.5: 2D Particle system with neighbour interactions, iteration 60

## 4.2 Implement a Particle System in 2D with collisions (A box with a small window)

For the second task we generate in random positions 70 particles, all of them are located at the left side of the box img. 4.6.

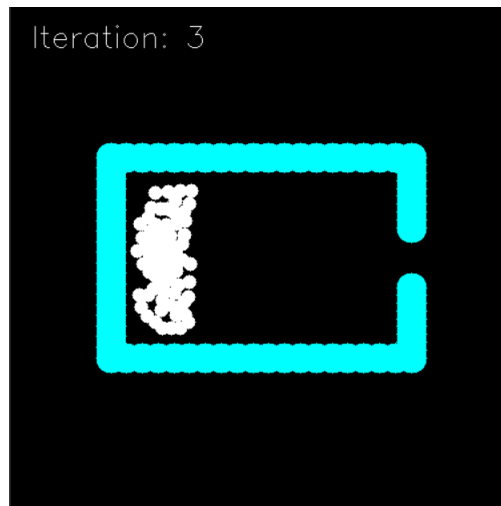


Figure 4.6: 2D Particle collisions (A box with small window), iteration 3

As you can see in each iteration looks like every particle moves to the right looking for the exit img. 4.7.

#### 4.2. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW)20

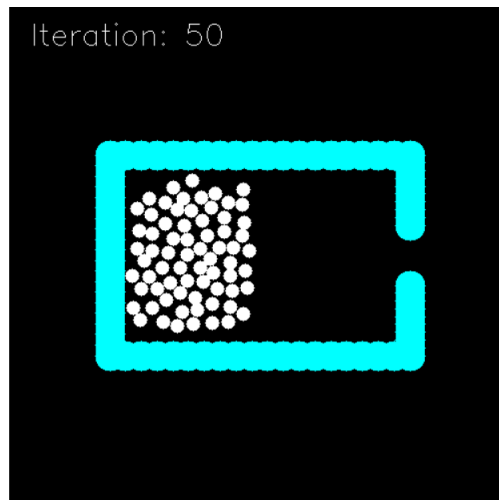


Figure 4.7: 2D Particle collisions (A box with small window), iteration 50

And this behaviour keeps going, but now looks like the particles are moving specifically to the exit, looks like they are moving in the  $y$  axis to the middle of the box 4.8.

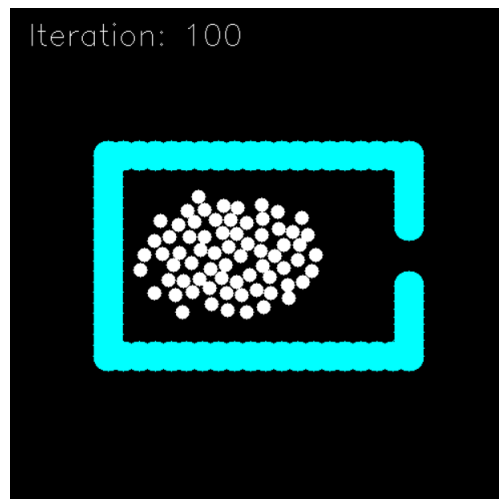


Figure 4.8: 2D Particle collisions (A box with small window), iteration 100

In the img. 4.9 we have the same behaviour but in the img. 4.10 as you can see some particles found the exit and the rest of them start following that particle.

4.2. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW)21

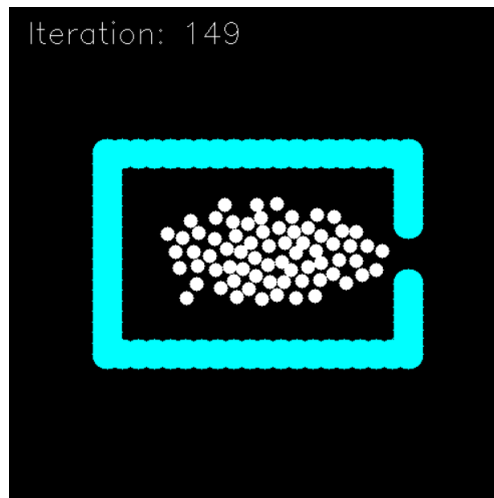


Figure 4.9: 2D Particle collisions (A box with small window), iteration 149

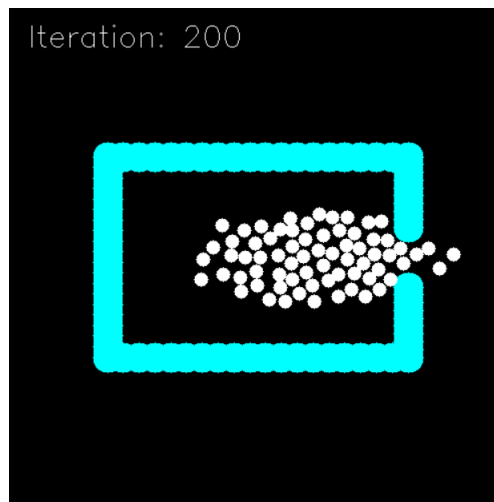


Figure 4.10: 2D Particle collisions (A box with small window), iteration 200

### 4.3 Implement a Particle System in 2D with collisions (A box with a small window and an obstacle)

We have a very similar behaviour than in the previous task, the particles moves from left to right  
img. 4.11

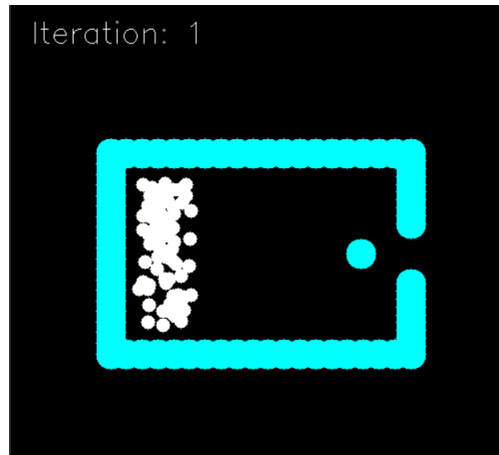


Figure 4.11: 2D Particle collisions (A box with small window and obstacle), iteration 1

And again looks like all the particles starts moving in the y axis too img. 4.12.

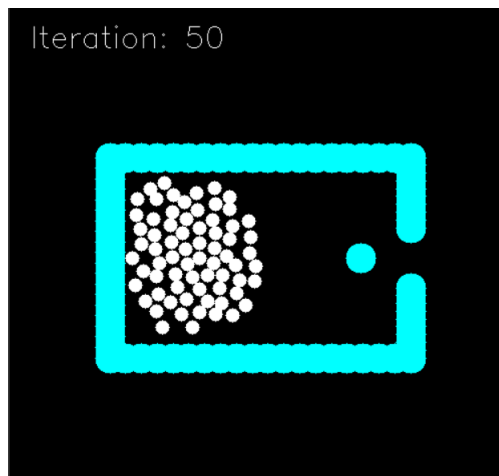


Figure 4.12: 2D Particle collisions (A box with small window and obstacle), iteration 50

#### 4.3. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW AND AN OBST

But now the particles found an obstacle they must surround that obstacle in order to move to the exit

4.13.

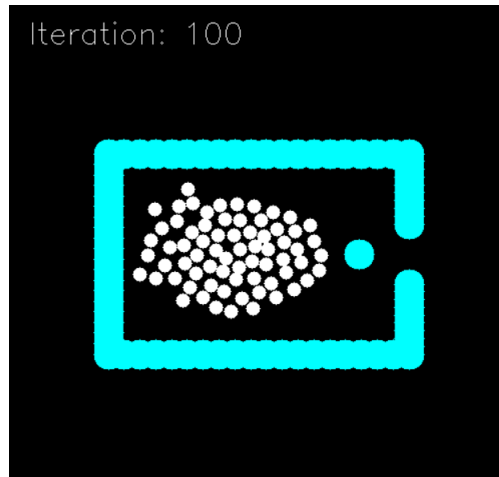


Figure 4.13: 2D Particle collisions (A box with small window and obstacle), iteration 100

And as we described previously, the particles surrounded that obstacle and continue moving to the exit

4.14.

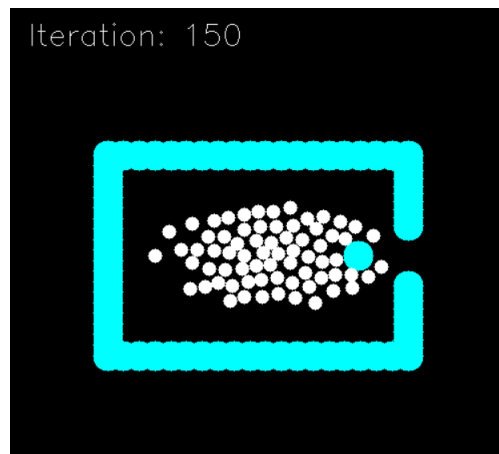


Figure 4.14: 2D Particle collisions (A box with small window and obstacle), iteration 150



#### 4.3. IMPLEMENT A PARTICLE SYSTEM IN 2D WITH COLLISIONS (A BOX WITH A SMALL WINDOW AND AN OBST

Looks like the particles goes to the exit in pairs and the exit of that particles is ordered 4.15.

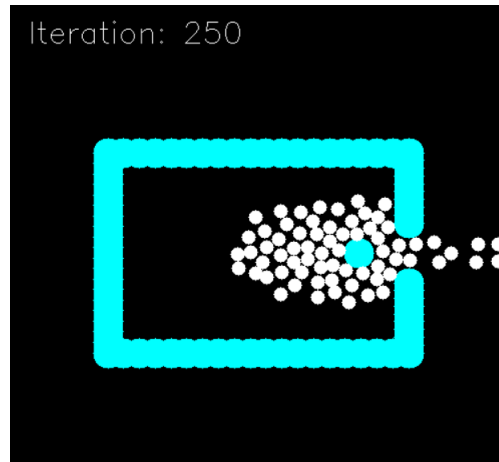


Figure 4.15: 2D Particle collisions (A box with small window and obstacle), iteration 250

## Conclusions

This practice has been very interesting at least for me, because we used some physic concepts to simulate some behaviours that we have in the real world. The first task was related to interaction, here every particle interacts with their neighbors (at least the up, down, left, right neighbour) and we get something very similar to a wave for example when we throw a rock in water. The other two tasks used the Newton's first and second law to describe the behaviour of each particle, but as we are using a 2-dimensional system we used vectors to modelate the behaviour of position, force and speed.

As a commentary I believe that provide us a template about how something works to make the practices is good but to make easier to understand the code I suggest to use more descriptive variable names or at least add a comment in the code with a brief description about what is the objective of a variable.

## Bibliography

- [1] William T. Reeves *Particle systems - A technique for modeling a Class of Fuzzy Objects*. Accessed on November 7th, 2021. Available online: <https://www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/fuzzyParticles.pdf>