**Exercise 1**  
  
Consider a scheme similar to Homework 3, Part a  
where M systems are subject to a series of N attacks. A system is discarded as "unsecure" if it reaches a penetration score of P **before** reaching, instead, a security score of S. Simulate and represent the probabilities of a system being discarded, for various values of P, example: P = k\*10 (k=2,...,10), conditional on the 3 cases for S: S = 20, S = 60, S = 100 (or any other value of S of your choice that you find useful to explore (it could be a user parameter).

SOURCECODE JS

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<link rel="icon" href="/assets/img/favicons/favicon.ico" />

<script src="/assets/static/js/Histogram.js"></script>

<script src="/assets/static/js/ScenarioSimulation.js"></script>

<script src="/assets/static/js/Chart.h6.js"></script>

<script src="/assets/static/js/ResizableRectangle.js"></script>

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap" rel="stylesheet">

<title>Homework 06</title>

<style>

html,

body,

#flexDiv {

font-family: 'Roboto', sans-serif;

width: 100%;

height: 100%;

overflow: hidden;

margin: 0;

padding: 0;

}

#flexDiv {

display: flex;

flex-direction: column;

}

#inputDiv {

width: 100%;

height: 5%;

background-color: black;

border-bottom: 2px solid #FBFAF5;

display: flex;

justify-items: start;

padding-top: 10px;

padding-bottom: 10px;

min-height: 40px;

}

#canvasDiv {

width: 100%;

height: 95%;

}

label {

color: #FBFAF5;

}

input {

font-family: 'Roboto', sans-serif;

width: 70px;

text-align: center;

border-radius: 7%;

height: 32px;

}

canvas {

background-color: black;

}

.inputs {

margin-right: 5px;

height: 38px;

}

.button {

font-family: 'Roboto', sans-serif;

cursor: pointer;

outline: 0;

color: #fff;

background-color: #0d6efd;

border-color: #0d6efd;

display: inline-block;

font-weight: 400;

line-height: 1.5;

text-align: center;

border: 1px solid transparent;

padding: 6px 12px;

font-size: 16px;

border-radius: .25rem;

transition: color .15s ease-in-out, background-color .15s ease-in-out, border-color .15s ease-in-out, box-shadow .15s ease-in-out;

}

.button:hover {

color: #fff;

background-color: #0b5ed7;

border-color: #0a58ca;

}

</style>

</head>

<body>

<div id="flexDiv">

<div id="inputDiv">

<div class="inputs">

<label for="M">M: </label>

<input class="input short-input" type="text" id="M" min="10" max="1000" value="100">

</div>

<div class="inputs">

<label for="N">N: </label>

<input class="input short-input" type="text" id="N" min="10" max="10000" value="1000">

</div>

<div class="inputs">

<button class="button" onclick="toggleLines(ResizableRectangle1, ResizableRectangle2, ResizableRectangle3)">Toggle White Lines</button>

</div>

<div class="inputs">

<button class="button" onclick="simulate()">Simulate!</button>

</div>

</div>

<div id="canvasDiv">

<canvas id="myCanvas" oncontextmenu="return false"></canvas>

</div>

</div>

<script>

"use strict";

const labelDateTime = document.getElementById("labelDateTime");

const myCanvas = document.getElementById("myCanvas");

const ctx = myCanvas.getContext("2d");

const inputM = document.getElementById("M");

const inputN = document.getElementById("N");

let simulating = false;

myCanvas.style.width = '100%';

myCanvas.style.height = '100%';

myCanvas.width = window.innerWidth

myCanvas.height = window.innerHeight;

ctx.transform(1, 0, 0, -1, 0, myCanvas.height);

window.addEventListener('resize', () => {

myCanvas.style.width = '100%';

myCanvas.style.height = '100%';

myCanvas.width = window.innerWidth

myCanvas.height = window.innerHeight;

ctx.transform(1, 0, 0, -1, 0, myCanvas.height);

});

ResizableRectangle.initialState.rectangle = new Rectangle(1, 1, 400, 300);

ResizableRectangle.initialState.ctx = ctx;

ResizableRectangle.initialState.outlineColor = "Orange";

ResizableRectangle.initialState.lineWidth = 1;

ResizableRectangle.initialState.lineDash = [1, 1];

const ResizableRectangle1 = new ResizableRectangle(ResizableRectangle.initialState);

ResizableRectangle.initialState.rectangle = new Rectangle(500, 1, 400, 300);

ResizableRectangle.initialState.ctx = ctx;

ResizableRectangle.initialState.outlineColor = "Purple";

ResizableRectangle.initialState.lineWidth = 1;

ResizableRectangle.initialState.lineDash = [1, 1];

const ResizableRectangle2 = new ResizableRectangle(ResizableRectangle.initialState);

ResizableRectangle.initialState.rectangle = new Rectangle(1000, 1, 400, 300);

ResizableRectangle.initialState.ctx = ctx;

ResizableRectangle.initialState.outlineColor = "Yellow";

ResizableRectangle.initialState.lineWidth = 1;

ResizableRectangle.initialState.lineDash = [1, 1];

const ResizableRectangle3 = new ResizableRectangle(ResizableRectangle.initialState);

let c1, c2, c3;

function drawScene() {

ctx.clearRect(0, 0, myCanvas.width, myCanvas.height);

ResizableRectangle1.drawChartNormal(c1);

ResizableRectangle2.drawChartNormal(c2);

ResizableRectangle3.drawChartNormal(c3);

}

function simulate() {

//s1 = new ScenarioSimulation(inputM.value\*1, 50, 0.9, -1, 1, 25, 10, (val, i) => val);

c1 = new Chart().Instance(inputN.value \* 1, inputM.value \* 1, 1, yFunc, 10, -20);

//console.log(c1.coordinates);

c2 = c1.getShallowCopy(10, -60);

c3 = c1.getShallowCopy(10, -100);

setInterval(drawScene, 5);

}

function yFunc(vals, trajectory, x) {

if (x <= 0) return 0;

if (Math.random() < 0.5) {

return vals[trajectory][x - 1] + 1;

}

else {

return vals[trajectory][x - 1] - 1;

}

}

function toggleLines(r1, r2, r3){

r1.whiteToggle = !r1.whiteToggle;

r2.whiteToggle = !r2.whiteToggle;

r3.whiteToggle = !r3.whiteToggle;

}

</script>

</body>

</html>

**The result is….**

