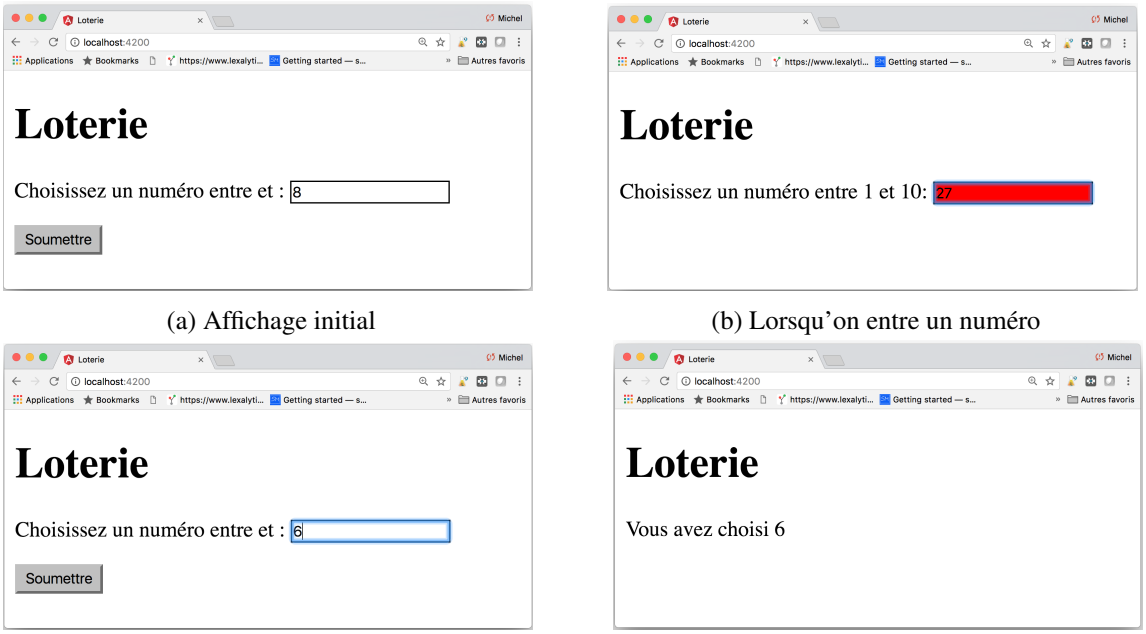

(a) Affichage initial

(b) Lorsqu'on entre un numéro

(c) Numéro invalide

(d) Après soumission

FIGURE 1 – Application de loterie

1 Plateforme MVC sur le client (4 points)

On vous demande d'écrire une application de loterie très simple. Dans un premier temps vous allez créer un client en utilisant la plateforme Angular. Lorsque nous chargeons l'application dans le navigateur, nous avons un affichage tel qu'illustré à la figure 1a. Notez qu'une valeur choisie aléatoirement a déjà été pré-sélectionnée dans la boîte d'input. On peut bien sûr modifier cette valeur. Lorsque le nombre est invalide, le fond apparaît en rouge (1b) et le bouton *Soumettre* disparaît et réapparaît lorsque le nombre est à nouveau valide (1c). Lorsqu'on clique sur le bouton *Soumettre*, l'affichage nous indique le nombre qu'on a choisi, tel qu'illustré à la figure 1d.

Fournissez le contenu des trois fichiers suivants : *app.component.ts*, *app.component.html* et *app.component.css*. Vous supposerez que le contenu du fichier *index.html* est le suivant :

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Loterie</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <app-root>Loading...</app-root>
  </body>
</html>
```

app.component.css :

```
.invalid { background-color: red}
input { border: solid black 1pt; }
```

app.component.html :

```
<h1> Loterie </h1>
<p *ngIf="!submitted">
  Choisissez un numero entre {{min}} et {{max}}:
  <input [(ngModel)]="selection" (ngModelChange)=validate() [class.invalid]="invalid">
</p>
<p *ngIf="selection && !submitted && !invalid">
    <button (click)="onSelection()"> Soumettre</button></p>
<p *ngIf="submitted"> Vous avez choisi {{selection}}</p>
```

app.component.ts :

```
import { Component } from '@angular/core';

  const MIN = 1;
  const MAX = 10;

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  private selection: number;
  private submitted: boolean;
  private invalid: boolean;

  public ngOnInit(): void{
    this.submitted = false;
    this.selection = this.generateNumber(MIN,MAX);
  }

  public onSelection(): void{
    this.submitted = true;
  }

  public validate(): void{
    this.invalid = !(MIN < this.selection && this.selection <= MAX);
  }

  public generateNumber(min: number, max: number): number {
    return Math.floor(Math.random() * max + min);
  }
}
```

(a) Le tirage a été réalisé et on a gagné

(b) Le tirage a été réalisé et on a perdu

(c) Nombre maximal de connexion atteint

FIGURE 2 – Application de loterie avec utilisation de *socket.io*

2 Socket.io (5 points)

Vous allez maintenant compléter l'application que vous avez élaborée à la question précédente, en y ajoutant des communications par le biais de websockets, en utilisant la bibliothèque *socket.io*. L'application se comporte de manière identique à ce qui a été demandé à la question précédente, avec la différence suivante.

Lorsque 10 clients sont connectés, le tirage est effectué (numéro choisi au hasard), le serveur communique avec les clients pour leur indiquer si leur numéro est gagnant ou non (l'affichage correspond alors à ce qui est illustré aux figures 2a et 2b). Attention : il ne leur dit pas quel est le numéro gagnant. Aucun autre client supplémentaire ne pourra alors participer au tirage : si on essaie de se brancher sur l'application, on aura alors l'affichage tel qu'illustré à la figure 2c. Bien sûr, si un client se déconnecte avant d'avoir soumis son numéro, sa place pourra alors être occupée par un autre client qui s'ajoute. Un client qui s'est déjà fait refuser sa connexion n'est pas avisé lorsqu'une connexion se libère.

Fournissez les nouvelles versions des fichiers *app.component.ts*, *app.component.html*, ainsi que le code sur le serveur. Le conteneur *Map()*, décrit brièvement en annexe, pourrait vous être utile.

app.component.html :

```
<h1> Loterie </h1>
<p *ngIf="!submitted && !maximum">
  Choisissez un numero entre 1 et 10:
  <input [(ngModel)]="selection" (ngModelChange)=validate() [class.invalid]="invalid">
</p>
<p *ngIf="selection && !submitted && !invalid">
```

```
    <button (click)="onSelection()"> Soumettre</button></p>
<p *ngIf="submitted && winningNumber === undefined"> Vous avez choisi {{selection}}</p>
<p *ngIf="gagnant"> Vous avez gagne</p>
<p *ngIf="perdant"> Vous avez perdu</p>
<p *ngIf="maximum"> Nombre maximum de participants atteint. Desole. </p>
```

app.component.ts :

```
import { Component } from '@angular/core';
import * as io from 'socket.io-client';
const MIN = 1;
const MAX = 10;

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  private selection: number;
  private submitted: boolean;
  private invalid: boolean;
  private socket: io.Socket;
  private gagnant: boolean;
  private perdant: boolean;
  private maximum: boolean;

  public ngOnInit(){
    this.socket = io('http://localhost:8000');
    this.socket.on('maximum',(m: string)=>{
                                this.maximum = true;
                                this.invalid = true;
                              });
    this.submitted = false;
    this.gagnant = false;
    this.perdant = false;
    this.maximum = false;
    this.selection = this.generateNumber(MIN,MAX);
  }

  public onSelection(){
    this.submitted = true;
    this.socket.emit('selection',this.selection);
    this.socket.on('result',(n: number)=>{
                              this.socket.disconnect();
                              this.gagnant = (n == this.selection);
                              this.perdant = !this.gagnant});
  }

  public validate(){
    this.invalid = !(MIN <= this.selection && this.selection <= MAX);
  }

  public generateNumber(min: number, max: number): number {
    return Math.floor(Math.random() * max + min);
  }
}
```

serveur :

```
let app = require('express')();
let http = require('http').Server(app);
let io = require('socket.io')(http);
let port = process.env.PORT || 8000;


let connected = new Map();
let maxNumberConnections = 4;

function allSubmitted(list){
    for (item of list.values()){
        if (!item) return false;
    }
    return true;
}

io.on('connection', function(socket){
    connected.set(socket.id,false);

    if (connected.size > maxNumberConnections){
        socket.emit('maximum','');
        connected.delete(socket.id);
    }
    else {
        socket.on('disconnect', function(){
            connected.delete(socket.id);
        });

        socket.on('selection',function(n){
            console.log(n);
            connected.set(socket.id,true);

            if (connected.size === maxNumberConnections &&
                allSubmitted(connected)){
                connected.clear();
                let winning = Math.floor(Math.random()*10+1);
                console.log(winning);
                io.emit('result', winning);
            }
        });
    }
});
```

3 Service Web (4 points)

Nous allons maintenant nous intéresser à une implémentation de notre loterie sous forme de service Web REST. Voici le code qui implémente ce service :

```
1  let app = require('express')();
2  let port = process.env.PORT || 8000;
3  let express = require('express');
4  let rest = require('restler');
5  let bodyparser = require('body-parser');
6
7  let participants = new Map();
8  let maxParticipants = 2;
9  let winningNumber;
10 let closed = false;
11
12 function generateId(){
13     return Math.floor(Math.random()*100000);
14 }
15
16 function processDraw(){
17     if (participants.size === maxParticipants){
18         closed = true;
19         winningNumber = 5;
20     }
21 }
22
23 app.use('/loto/*',bodyparser.json({ type: 'application/json' }));
24
25 app.use('/loto/selection/*',function(req,res,next){
26     if (closed){
27         res.json({success:false, error:"loto is closed"});
28     }
29     else {
30         next();
31     }
32 });
33
34 app.get('/loto/result/:id',function(req,res){
35     if (closed) {
36         res.json({closed: true,
37                   winning: winningNumber === participants.get(Number(req.params.id))})
38     }
39     else {
40         res.json({closed: false});
41     }
42 });
43
44 app.get('/loto/selection/:id',function(req,res){
45     res.json({selection: participants.get(Number(req.params.id))});
46 });
47
48
49
```

```
50  app.post('/loto/selection/:number',function(req,res){
51      if (closed){
52          res.json({closed: true});
53      }
54      else {
55          let newId = generateId();
56          participants.set(newId,Number(req.params.number));
57          processDraw();
58          res.json({closed:closed, id:newId});
59      }
60  });
61
62  app.delete('/loto/selection/:id',function(req,res){
63      if (participants.has(Number(req.params.id))){
64          participants.delete(Number(req.params.id));
65          res.json({success:true});
66      }
67      else {
68          res.json({succes:false, error:"id does not exist"});
69      }
70  });
71
72  app.put('/loto/selection/:id/:number',function(req,res){
73      if (participants.has(Number(req.params.id))){
74          participants.set(Number(req.params.id),Number(req.params.number));
75          res.json({success:true});
76      }
77      else {
78          res.json({success:false, error:"id does not exist"});
79      }
80  });
81
82  app.listen(port, function(){
83    console.log('listening on *:' + port);
84  });
```

a) (3 points) Expliquez comment fonctionne ce service Web, du point de vue d'un client qui y fait appel par le biais de requêtes HTTP. En particulier, expliquez comment on participe à la loterie, en décrivant toutes les fonctionnalités offertes par ce service, ainsi que la manière de les utiliser.

Pour participer au tirage, on fait un POST à la ressource /loto/selection/<numero>, où <numéro> est le numéro choisi pour le tirage. Le serveur nous retourne un identificateur, qui pourra être utilisé pour se retirer du tirage (requête DELETE), pour changer de numéro (requête PUT avec notre identificateur et un nouveau numéro) ou pour savoir le numéro que nous avons soumis au tirage (requête GET avec notre identificateur). Pour savoir si on a gagné, on fait un GET sur /loto/result avec l'identificateur. Si le tirage a été fait, le serveur nous indique si on a gagné.

b) (1 point) Expliquez le code aux lignes 25-32, et pourquoi la méthode *use* est utilisée pour ce middleware.

Ce middleware est appelé pour toutes les requêtes. Il permet de retourner tout de suite un message au client lorsque le tirage est terminé. Sinon, on poursuit pour traiter la requête.

4 Infographie avec three.js (5 points)

Soit le code suivant, qui implémente une application infographique (il y aurait bien sûr beaucoup de remaniement à faire dans ce code, mais ce n'est pas ce qui nous intéresse ici) :

index.html :

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <title>three.js example</title>
                <meta charset="utf-8">
                <meta name="viewport"
                      content="width=device-width,
                      user-scalable=no,
                      minimum-scale=1.0,
                      maximum-scale=1.0">
        </head>
        <body>
                <script src="js/three.js"></script>
                <script src="js/Detector.js"></script>
                <script src="app.js"></script>
        </body>
</html>
```

app.js :

```
let container;
let camera, scene, renderer;
let plane, cube;
let mouse, raycaster, isShiftDown = false;
let cubeGeometry, cubeMaterial;
let objects = [];

const SHIFT = 16;
const LEFT_ARROW = 37;
const UP_ARROW = 38;
const RIGHT_ARROW = 39;
const DOWN_ARROW = 40;

const CAMERA_POS = [new THREE.Vector3(-800,500,0),
                          new THREE.Vector3(-500,500,500),
                          new THREE.Vector3(0,500,800),
                          new THREE.Vector3(500,500,500),
                          new THREE.Vector3(800,500,0),
                          new THREE.Vector3(500,500,-500),
                          new THREE.Vector3(0,500,-800),
                          new THREE.Vector3(500,500,-500)];
let index_camera_position;

init();
render();
```

```
31 function init() {
32     container = document.createElement('div');
33     document.body.appendChild(container);
34 
35     index_camera_position = 0;
36     camera = new THREE.PerspectiveCamera(45, window.innerWidth/window.innerHeight, 1, 10000);
37     camera.position.copy(CAMERA_POS[0]);
38     camera.lookAt(new THREE.Vector3());
39 
40     scene = new THREE.Scene();
41 
42     cubeGeometry = new THREE.BoxGeometry(50, 50, 50);
43     cubeMaterial = new THREE.MeshLambertMaterial(
44             {color: 0xfeb74c,
45              map: new THREE.TextureLoader().load( "./textures/square-outline.png")});
46 
47     let size = 200;
48     let step = 50;
49     let geometry = new THREE.Geometry();
50 
51     for (let i = -size; i <= size; i += step) {
52         geometry.vertices.push(new THREE.Vector3(-size, 0, i));
53         geometry.vertices.push(new THREE.Vector3(size, 0, i));
54         geometry.vertices.push(new THREE.Vector3(i, 0, -size));
55         geometry.vertices.push(new THREE.Vector3(i, 0, size));
56     }
57 
58     let material = new THREE.LineBasicMaterial({color: 0x000000,
59                                                 opacity: 0.2,
60                                                 transparent: true});
61     let line = new THREE.LineSegments(geometry, material);
62     scene.add(line);
63 
64     raycaster = new THREE.Raycaster();
65     mouse = new THREE.Vector2();
66 
67     geometry = new THREE.PlaneBufferGeometry(1000, 1000);
68     geometry.rotateX(-Math.PI / 2);
69 
70     plane = new THREE.Mesh(geometry, new THREE.MeshBasicMaterial({visible: false}));
71     scene.add(plane);
72 
73     objects.push(plane);
74 
75     let ambientLight = new THREE.AmbientLight(0xffffff);
76     scene.add(ambientLight);
77 
78     renderer = new THREE.WebGLRenderer({antialias: true});
79     renderer.setClearColor(0xf0f0f0);
80     renderer.setPixelRatio(window.devicePixelRatio);
81     renderer.setSize(window.innerWidth, window.innerHeight);
82     container.appendChild( renderer.domElement );
83 
84     document.addEventListener('mousedown', onMouseDown, false);
85     document.addEventListener('keydown', onKeyDown, false);
86 }
```

```
87
88  function onMouseDown(event) {
89      event.preventDefault();
90      mouse.set((event.clientX / window.innerWidth)*2 - 1,
91               -(event.clientY/window.innerHeight)*2 + 1);
92      raycaster.setFromCamera(mouse, camera);
93      let intersects = raycaster.intersectObjects(objects);
94
95      if (intersects.length > 0) {
96              let intersect = intersects[0];
97              let cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
98              cube.position.copy(intersect.point).add(intersect.face.normal);
99              cube.position.divideScalar(50).floor().multiplyScalar(50).addScalar(25);
100             scene.add(cube);
101             objects.push(cube);
102         render();
103     }
104 }
105
106 function onKeyDown(event){
107     if (event.keyCode === RIGHT_ARROW || event.keyCode === LEFT_ARROW){
108             if (event.keyCode === RIGHT_ARROW){
109                 index_camera_position = (index_camera_position + 1) % CAMERA_POS.length;
110             }
111             else {
112                 index_camera_position =
113                    (index_camera_position + CAMERA_POS.length - 1) % CAMERA_POS.length;
114             }
115             camera.position.copy(CAMERA_POS[index_camera_position]);
116             camera.lookAt(new THREE.Vector3());
117             render();
118     }
119     if (event.keyCode === UP_ARROW || event.keyCode === DOWN_ARROW){
120             if (event.keyCode === UP_ARROW){
121             camera.position.y += 50;
122         }
123         else {
124             camera.position.y -= 50;
125         }
126         camera.lookAt(new THREE.Vector3());
127         render();
128      }
129 }
130
131 function render() {
132     renderer.render(scene, camera);
133 }
```

Décrivez l'application infographique implémentée par ce code. Décrivez ce qui est affiché et ce qui se produit à chaque action possible de l'utilisateur.

Elle crée une grille sur le sol. Quand on clique sur une case de la grille, un cube est ajouté sur cette case. Quand on clique sur la surface d'un cube, un autre cube est ajouté en le juxtaposant à cette surface. Lorsqu'on clique sur flèche droite ou gauche, la caméra se déplace de 45 degrés dans la direction spécifiée. Lorqu'on clique sur la flèche haut ou bas, la caméra monte ou descend, respectivement.

5 Intégration continue (1 point)

Décrivez trois facteurs d'importance majeure pour assurer le succès d'un développement de logiciel en intégration continue, dans une approche agile.

1) La présence de tests
2) Des commits de granularité très fine
3) Du code de bonne qualité

6 Base de données NoSQL (1 point)

Soit la base de données en MongoDB représentant des restaurants, où chaque document respecte la structure illustrée par l'exemple suivant :

```
{
"address": {
   "building": "1007",
   "coord": [ -73.856077, 40.848447 ],
   "street": "Morris Park Ave",
   "zipcode": "10462"
},
"borough": "Bronx",
"cuisine": "Bakery",
"score":  10,
"name": "Morris Park Bake Shop",
"restaurant_id": "30075445"
}
```

a) Quelle est la requête pour obtenir la liste de tous les restaurants situés dans le Bronx ?

document.find({"borough" :"Bronx})

b) Quelle est la requête pour obtenir les cinq premiers restaurantstrouvés qui ont un score supérieur à 5 ?

document.find({"score" :{$gt : 5}}).limit(5)

7 Question bonus (0,5 point)

De quel pays proviennent les championnes du monde de Curling en 2017 ? Choisissez une réponse parmi les suivantes : Canada, États-Unis, Russie, Suède, France, Danemark, Suisse, Finlande, Allemagne, Maroc.

Canada

Annexe - conteneur Map()

L'objet Map représente un dictionnaire, autrement dit une carte de clés/valeurs. N'importe quelle valeur valable en JavaScript (que ce soit les objets ou les valeurs de types primitifs) peut être utilisée comme clé ou comme valeur.
On peut savoir le nombre d'items qu'il contient par le biais de son attribut *size*.
Son interface est composée des méthodes suivantes :

clear() Supprime tous les items qu'il contient.

set(clé,item) Associe un item à une clé.

delete(clé) Retire l'item associé à la clé.

get(clé) Permet d'obtenir l'item associé à la clé.

has(clé) Indique si la clé existe dans le Map.