1 Angular 2:

1.1 Communication à travers les components :

Parent vers enfant -> Input ou ViewChild/ViewChildren example:

@ViewChildren('btnJoueur') boutonsJoueur: QueryList<ElementRef>;

Retourne une QueryList<typeDuViewChildren> de tous les components/éléments HTML avec la template variable 'btnJoueur'. Pour utiliser la QueryList on peut utiliser .toArray(). Lorsque le viewChildren est utilisé sur un élément HTML, le type est ElementRef, celui-ci possède 1 attribut(nativeElement) qui représente l'élément HTML. Example, this.boutonJoueur.toArray[0].nativeElement.classList;

Example template variable: <h1 #btnJoueur>je ne suis pas un vrai btn</h1>

@ViewChild(ChatLobbyComponent) chat: ChatLobbyComponent;

Retourne une instance du component en question (Un appel de VC possible seulement) ex:

- @ViewChild(ChatLobbyComponent) chatL: ChatLobbyComponent;
- @ViewChild(ChatComponent) chat: ChatComponent;

Dans ce cas, chat sera undefined et chatL aura la bonne instance du composant, Solution remplaer ChatComponent dans le ViewChild par une template Variable.

Enfant vers parent -> EventEmitter

Example:

Début de la classe(Initialisation) dans l'enfant :

@Output() onClose = new EventEmitter();

Émettre l'événement dans l'enfant:

this.onClose.emit():

Gérer l'événement dans le parent :

<home>(onClose) = 'this.fermerFenetre()'<`home> ← balise de
création de l'enfant

1.2 Creation d'un component :

Dans le template:

() => "sort" du component(events et ngModel)

[] => "entre" dans le component(input et ngModel)

```
* = > directives angular
@ => decorators
Attributs dynamiques dans le template:
Exemple:
class = "case" + i" VS [class] = ""case" + i"
Le premier cas la classe va etre "case + i" tandis que le deuxième changera selon la
valeur de i(case1, case2, case3).
Gabarit de base d'un composant:
import { GameComponent } from '../Game/game.component';
... (import tous les components + services necessaires)
import * as io from 'socket.io-client';
@Component({
      Selector: 'kebab-case',
       Template: `
             *Html file * On peut aussi utiliser TemplateURL.
      Providers: [ VerificatorService, ... (autres services)]
      })
      Permet d'établir que le composant pourra utiliser ledit service
      export class ControllerComponent implements AfterViewInit
        // Initialisation des ViewChild + emitters (absent dans l'exemple)
        @ViewChild(GameComponent) game: GameComponent;
        @ViewChild('home') home: HomeComponent;
        // Initialisation des variables
        ouvrirMenu: boolean:
        nomJoueur: string;
        constructor(private verif: verificatorService);//la position du constructor est
      importante!: le services sont initialises ici
       // Functions
1.3 Specialite Angular nglf, ngFor, ngModel, ...:
nglf : affiche un élément html si la condition est respectée :
      <div *nglf='this._showTimer'>
ngFor : pour chaque élément d'une array, va créer un objet html:
       <span *ngFor = 'let rangee of this.rangees; let i = index' class = "grid-line">
                     <input type = "text" [value] = 'this.calculateValue(i)'
```

```
readonly class = "grid-name grid-case-name">
</span>
```

Pour chaque élément de this.rangees, un span va être créé avec un input à l'intérieur.

ngModel: Data binding entre un élément html et les données:

Example:

<input #inputGrid type = "text" [(ngModel)] = 'this.rangees[i].colonnes[j].valeur' >

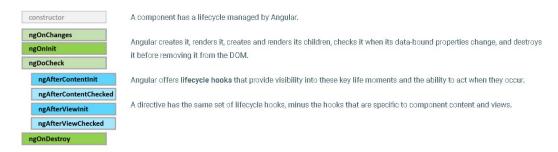
La boite d'input est maintenant lié avec la bonne case dans l'array de colonnes (TWO-WAY DATA BINDING)

Interpolation: Permet d'afficher une variable du code dans le template html

Example:

This.texte = 'Bonjour'; (dans la classe)
<h1> {{this.texte}} </h1> (dans le template html)

1.4 Angular lifecycle hooks



Important hooks:

ngOnInit: apelle lors de l'initialisation du composant ngAfterViewInit: apelle lorsque toutes les éléments du template (aussi appelé la view) ont été initialisé

ngAfterViewChecked: appelle après ngAfterViewInit et à tous les changements de template détectés

ngOnDestroy: appelle juste avant la destruction du composant