Commencé le	mardi 8 mars 2022,
État	Terminé
Terminé le	mardi 8 mars 2022
Temps mis	
Note	**16,57** sur 25,00 (**66**%)

Description

LISEZ CES INSTRUCTIONS AVANT DE COMMENCER L'EXAMEN

- L'examen possède 11 questions notées sur un total de 25 points.
- La note partielle associée à chaque question est marquée à gauche de la question.
- Certaines questions demandent d'ajouter une justification à votre réponse. Une bonne réponse sans justification valide ne sera pas acceptée.
- **Vous avez une seule tentative pour l'examen! Ne soumettez pas votre tentative à moins d'être 100% sûr(e) d'avoir terminé l'examen ! La soumission sera automatique à la fin de la période.**
- **En cas de doute sur le sens d'une question, faites une supposition raisonnable, énoncez-là clairement dans votre réponse et poursuivez. Une "question" vide est disponible sur la première page d'examen si vous avez besoin de plus de place ou pour des questions spécifiques.**
- **Vous avez accès à une copie des notes de cours PDF sur l'instance Moodle et droit à 1 feuille recto-verso (imprimée ou manuscrite) comme documentation.**

Bon travail!

Question **1**

Terminer

Non noté

Cette question est un espace dédié à vos commentaires ou questions sur l'examen. Nous ne répondons à aucune question pendant l'examen.

Priorisez de mettre vos commentaires et/ou hypothèses directement dans la question spécifique.

ok

Question 11) a)
Pour vérifier que les tests sont toujours de bonne qualité.

On les fait rouler, c'est possible qu'ils passent toujours, mais qu'il ne soit plus d'aussi bonne qualité (possible qu'on doive un peut modifier les tests après une petites modification) Mais avant de modifier les tests, il faut les faire rouler pour quand même avoir une idée de ceux qui passent et ceux qui ne passent pas et pourquoi (aider à se remémorer le fonctionnement des tests peut aider à les coder ou a en ajouter des similaires). Voilà pourquoi j'ai aussi coché cette case

Question **2**

Terminer

Note de 2,50 sur 4,00

Vous êtes en charge du réusinage du chevalet ainsi que la fonctionnalité de manipulation de lettres au début du Sprint 2. Chaque tuile est représentée par l'interface suivante :

```
interface Tile {
    letter: string,
    value: number
}
```

Vous avez décidé de représenter chaque tuile sur votre chevalet par un Component : **TileComponent**. Voici l'implémentation du Component et son gabarait :

```
@Component({
  selector: 'app-tile',
  templateUrl: './tile.component.html',
  styleUrls: ['./tile.component.css']
})
export class TileComponent{
  @Input() tile : Tile;
}
```

Gabarit HTML :

```
<div id="container">
  <span id="letter">{{ tile.letter }}</span>
  <span id="value" >{{ tile.value }} </span>
</div>
```

Afin de prototyper rapidement le déplacement d'une lettre, vous avez implémenté les fonctions "*moveLeft*" et "*moveRight*" dans votre **RackService.** Assumez que le code est fonctionnel. Vous avez également la fonction "*getRack*" qui retourne votre chevalet. Voici une partie du service :

```
export class RackService {

  private tileRack: Tile[];
  getRack(): Tile[] { return this.tileRack; }

  moveRight() { ... }
  moveLeft() { ... }

}
```

Votre chevalet est représenté par **TileRackComponent**. Voici le Component et le gabarit que vous devez compléter :

```
export class TileRackComponent {

  tileRack: Tile[] = [];
  constructor(private rackService: RackService) {
    this.tileRack = this.rackService.getRack();
  }

  moveLeft() { this.rackService.moveLeft() }
  moveRight() { this.rackService.moveRight() }
}
```

Gabarit à compléter :

```
<!-- À COMPLÉTER -->
<div id="container">




</div>

<button (click)="moveLeft()">Gauche</button>
<button (click)="moveRight()">Droite</button>
```

a) Complétez le gabarit de **TileRackComponent** pour pouvoir afficher votre chevalet et pouvoir prototyper le déplacement. (2 points)

b) La valeur de tileRack est assignée seulement une fois dans le *constructeur* de **TileRackComponent** et nulle part ailleurs. Est-ce

que l'affichage de votre chevalet sera mis à jour si l'objet tileRack de **RackService** est modifié par la suite à travers les fonctions "*moveLeft*" et "*moveRight*" ? Justifiez votre réponse. (2 points)

a)

```
< div *ngFor="let tile of tileRack">
    < div> {{tile.letter}} </div>
    < div> {{tile.value}} </div>
 </div>
```

b) Oui, ce sera mis a jour automatiquement. La valeur sera automatiquement mise a jour étant donné que Angular re-render la vue de chaque component dynamiquement apres chaque modification.

Commentaire :
a) Il faut utiliser <app-tile> et lui passer la tuile en paramètre -1.

Question **3**

Partiellement correct

Note de 0,67 sur 1,00

Vous décidez de faire communiquer deux de vos Components à travers un Service partagé. Parmi les choix suivants, le ou lesquels sont des avantages architecturaux d'utiliser un Service partagé au lieu de l'utilisation des décorateurs @Input/@Output pour la communication entre 2 Components ?

- [ ] a. @Input/@Output permettent le passage de valeurs primitives, contrairement aux Services qui permettent la communication d'objets complexes.
- [ ] b. Les décorateurs nécessitent des variables publiques, ce qui brise le concept d'encapsulation, contrairement à l'utilisation d'un Service privé.
- [x] c. L'utilisation d'un Service diminue le couplage entre les Components. ✔
- [x] d. Les Services sont des singletons, contrairement aux Components. ✖
- [x] e. L'utilisation d'un Service élimine le besoin d'avoir un lien parent/enfant entre les Components. ✔

Votre réponse est partiellement correcte.

Vous avez sélectionné trop d'options.
Les réponses correctes sont :
L'utilisation d'un Service élimine le besoin d'avoir un lien parent/enfant entre les Components.,

L'utilisation d'un Service diminue le couplage entre les Components.

Question **4**

Terminer

Note de 2,00 sur 3,00

Quelle est la différence entre le serveur lancé lorsqu'on fait **npm start** dans le dossier **client** et le serveur lancé par la même commande dans le dossier **serveur** de votre projet ? Quel est le rôle de chacun de ses 2 serveurs? **Justifiez** votre réponse.

Bon alors npm start du client ****sans le serveur**** lance un acces a une page dans le navigateur, le client offre le html, le js et le css. On peut naviguer entre les pages de l'application tant qu'on ne nécessite pas impérativement d'utiliser le serveur (ex utilisation de socket pour créer des rooms peut bloquer la navigation sur la page du client)

Le npm start du serveur lance (dans notre projet) simplement l'écoute sur le port 3000. Ce serveur attent de recevoir des requet du client. Si je ne me trompe pas, on pourrait jouer au jeu juste avec curl + le serveur lancé (curl imiterait le client en envoyant toutes les requets qu'un utilisateur régulier enverrai avec le UI).

Note, j'utilise *****sans le serveur ou sans le client**** pour illustrer ce que chacun fait indépendamment de l'autre

Si je n'ai pas tous mes points pour cette questions, je serai triste, c'est que je me serai mal exprimé. Je me suis occupé du déploiement du serveur et du client dans notre équipe.

Commentaire :
Vous mélangez client et serveur. Le serveur **statique** dans le dossier client fournit des fichiers aux clients (navigateur), mais ne fait aucune gestion de navigation de page ou logique d'application. Le client ne peut pas "offrir" des fichiers, mais seulement les récupérer.

Question **5**

Incorrect

Note de 0,00 sur 1,00

Vous êtes responsables de l'implémentation de la logique du joueur virtuel pour le Sprint 2 et vous avez trouvé un service en ligne qui permet de générer des anagrammes. Vous voulez l'utiliser pour générer les placements possibles pour votre joueur virtuel. Cependant, le serveur du service n'utilise pas Socket.IO, mais plutôt l'implémentation native de WebSocket pour sa communication.

Pouvez-vous quand même utiliser ce service dans votre projet dans son état actuel ?

- a. Oui, Socket.IO utilise le protocole WebSocket pour communiquer, donc les 2 systèmes sont compatibles.
- b. Oui, mais vous ne pouvez pas utiliser les fonctionnalités supplémentaires que Socket.IO offre par-dessus WebSocket. ✖
- c. Non, votre serveur ne peut pas communiquer avec un autre serveur, seulement avec des clients.
- d. Non, le client et le serveur doivent utiliser Socket.IO pour une communication valide.
- e. Oui, mais vous devez contacter le service à travers votre client puisque WebSocket est un protocole client-serveur.

Votre réponse est incorrecte.

La réponse correcte est :
Non, le client et le serveur doivent utiliser Socket.IO pour une communication valide.

Question **6**

Correct

Note de 1,00 sur 1,00

Quelle est la différence entre Express, NodeJS et Socket.IO ?

- a. NodeJS est un environnement d'exécution à partir duquel on utilise les librairies Express et Socket.IO, pour une gestion à plus haut niveau des requêtes HTTP et de la communication WebSocket, respectivement. ✔
- b. NodeJS, Express et Socket.IO sont des librairies côté serveur pour la communication réseau.
- c. La librairie Socket.IO est nécessaire pour traiter le protocole WebSocket, mais Express est optionnelle pour un serveur utilsant NodeJS.
- d. La librairie Express est nécessaire pour traiter des requêtes HTTP, mais Socket.IO est optionnel pour un serveur utilsant NodeJS.

Votre réponse est correcte.

La réponse correcte est :
NodeJS est un environnement d'exécution à partir duquel on utilise les librairies Express et Socket.IO, pour une gestion à plus haut niveau des requêtes HTTP et de la communication WebSocket, respectivement.

Question **7**

Terminer

Note de 2,00 sur 4,00

Voici l'implémentation de la gestion du clavaradge dans plusieurs salles de votre projet.
Vous avez présentement 3 clients qui communiquent avec votre serveur : Client1, Client2 et Client3.
Client1 et Client3 sont présentement dans une partie de jeu et dans la même *Room* ayant le nom "123".

Voici la gestion de cet événement du côté serveur. La fonction *getRoomFromSocketId(socketId)* retourne l'identifiant de salle d'un socket quelconque :

```
socket.on('chatMessage', (message) => {
  const room = this.getRoomFromSocketId(socket.id);
  const chatMessage = { ...message, room: room }
  socket.broadcast.emit("chatMessage", chatMessage);
});
```

Voici également la gestion du message du serveur du côté client. L'attribut *roomId* représente le nom de la Room dans laquelle le client est présentement.

```
this.socketService.on('chatMessage', (chatMessage) => {
  if (this.roomId === chatMessage.room) {
    this.chatMessages.push(chatMessage);
  }
});
```

Client1 vient d'envoyer l'événement "chatMessage" avec l'objet **{ author: "Client1", message: "Allo"}** au serveur.
a) En fonction de la configuration donnée, qui recevera un message du serveur et **pourquoi** ?

b) Cette implémentation est fonctionnelle, mais n'est pas optimale. Donnez le problème avec l'implémentation et proposez une solution possible.

a) A cause du `broadcast, Client1, Client2 et Client3 recevront l'event. Le broadcast envoie le message a tous les clients.`

Par contre, seul les messages de client 1 et 3 seront affichés étant donné qu'ils respectent la condition du if`(this.roomId === chatMessage.room)`

`Ce n'est pas optimal comme traitement. Meme si c'est fonctionnel (Client2 ne verra pas de message affiché)`

b) Il faudrait modifier le premier bloque de code de la question pour simplement envoyer l'event aux socet ayant le bon ID (pour envoyer juste a client1 et client3). Puis supprimer la condition du if dans le deuxième bloque de code (c'est l'idée générale)

```
socket.on('chatMessage', (message) => {
  const room = this.getRoomFromSocketId(socket.id);
  const chatMessage = { ...message, room: room }
  socket.emit("chatMessage", chatMessage, socket.id); // suppression du broadcast
```

```
   socket.emit("chatMessage", chatMessage, this.getOpponentSocketId(socket.id)); // ajout de ligne cette fonction
donne l'id de l'opposant
```

// dans mon code je le verrai, mais nous pour emit on précisait directement les id des sockets concernés par le emit, donc le permier emit on l'envoie a l'emmeteur (socket.id) et l'autre a son partenair dans la salle (avec getOpponentSocketId(socket.id)) // dans mon code je fais ca sans probleme, mais la dans l'exam je n'ai pas les fonctions dont j'ai besoin NOTE: j'ai quand meme codé getOpponentSocketId plus bas

```
});
```

```
this.socketService.on('chatMessage', (chatMessage) => {
    this.chatMessages.push(chatMessage); // condition supprimée
});
```

```
getOpponentSocketId(string: emiterId): string {
```

```
      const room = this.getRoomFromSocketId(socket.id);
```

return (room.firstId = emiterId) ? room.secondId : room.firstId ; // en supposant que que room.firstId retourne un des deux id de socket dans la room et que room.secondId retourne l'autre

```
}
```

Commentaire :

a) Client1 ne reçoit pas le message. Broadcast n'envoi pas de mesage à l'émetteur original -1

b) La réponse est bonne, mais le code fourni est erroné et ne répond pas à la syntaxe de Socket.IO -1

Question **8**

Terminer

Note de 2,00 sur 3,00

Voici le constructeur du component **PlayAreaComponent** qui utilise la classe **GridService** ainsi que la configuration de ses tests unitaires.

```
constructor(private readonly gridService: GridService) {}

// play-area.component.spec.ts
beforeEach(async () => {
   gridSpy = jasmine.createSpyObj('GridService', ['placeLetter','drawGrid','changeSize']);
   TestBed.configureTestingModule({
       declarations: [PlayAreaComponent],
       providers: [{ provide: GridService, useValue: gridSpy }],
   }).compileComponents();
});
```

a) Est-ce que l'utilisation d'un SpyObj pourrait être nécessaire dans la configuration? **Justifiez** votre choix.

b) Sachant qu'aucune erreur n'est lancée dans la console pour ce Component et en vous basant sur l'objet **declarations** du module de tests, que pouvez-vous déduire du contenu du gabarit de PlayAreaComponent ?

a) Oui, en Jasmine un SpyObj est en réalité un mock (le SpyOn est un vrai spy), selon les définitions des notes de cours.

Donc utiliser un SpyObj au lieu d'utiliser le vrai `GridService permet de réduire les dépendances des tests et de rendre le tests plus unitaires disons.`

`b) Pour l'instant, PlayAreaComponent n'utilise pas grand chose d'autre qu'un GridService (c'est son seul provider nécessaire)`

Commentaire :

a) Parfait

b) Dans declarations on met plutôt les éléments présents dans le gabarit. Par exemple si on a une référence vers un autre component Angular, il faut l'indiquer pour éviter d'avoir des erreurs de HTML dans les tests

Question **9**

Correct

Note de 1,00 sur 1,00

Vous venez de terminer l'implémentation de l'intégration de la communication par sockets sur votre site web et vous êtes en mesure d'échanger des messages avec votre serveur.

Cependant, lors de l'exécution de vos tests, vous avez parfois un problème avec les tests des sockets qui n'arrivent pas à faire une connexion valide avec un serveur. Cette erreur se présente des fois lorsque vous lancez les tests sur votre machine, mais arrive toujours lorsque les tests sont exécutés par le pipeline automatisé de GitLab.

Quelle est la source d'erreur la plus probable dans cette situation ?

a. Les tests font des appels vers un vrai serveur Socket.IO. Ils réussissent lorsque vous laissez votre serveur local ouvert sur votre machine et échouent sur GitLab qui n'a pas de serveur de disponible. ✔

b. Le protocole WebSocket a besoin d'une connexion réseau pour fonctionner, ce qui n'est pas le cas sur la machine de pipeline.

c. WebSocket est un protocole instable qui fonctionne mal sur Linux et GitLab utilise toujours des serveurs Linux pour exécuter les pipelines.

d. Les tests font des appels vers un vrai serveur Socket.IO. La machine qui exécute les tests sur GitLab n'a pas la même addresse que votre machine locale et la connexion échoue toujours.

Votre réponse est correcte.

La réponse correcte est :
Les tests font des appels vers un vrai serveur Socket.IO. Ils réussissent lorsque vous laissez votre serveur local ouvert sur votre machine et échouent sur GitLab qui n'a pas de serveur de disponible.

Question **10**

Terminer

Note de 3,00 sur 4,00

Le code qui suit contient plusieurs défauts. Modifiez-le afin de corriger tous les défauts. Assurez-vous que votre code final respecte les standards de qualité et ne contienne pas de mauvaises odeurs.

Vous pouvez créer autant de nouvelles fonctions que nécessaire. Si vous désirez ajouter de nouveaux noms, ou en modifier, choisissez des noms le plus adéquats possible selon votre compréhension du code.

Il se peut aussi que votre code final contienne encore des mauvaises odeurs que vous ne pouvez corriger par manque d'information sur le contexte du code. Dans ce cas, décrivez-les par un court texte.

```
replaceLetters(a: Letter): boolean {
   let correct: boolean = isValidItem(a) ? true : false;
   if (correct === true) {
       let newLetter = this.letterBank.obtenirAleatoire(a);
       this.letters.foreach((v, i) => {
           if (this.items[i] === a) { this.items[i] = newLetter; }
       });
       return true;
   }
   else if (correct === false) {
       return false;
   }
}
```

```
replaceLetters(replacedLetter: Letter): boolean { // nom explicite de variable
   // suppression de code inutile
   if (idValidItem(replacedLetter)) { // suppression de code inutile
       const newLetter = this.letterBank.getRandom(a); // suppression de francais, const a la place du let
       this.letters.foreach((v, i) => { // je ne sais pas si on est obligé d'utiliser v et i pour que la boucle
intelligente comprenne, si ce n'est pas le cas, j'aurais utilisé value et index a la place, dans la vraie vie je
l'aurais testé rapido
           if (v === replacedLetter) this.items[i] = newLetter; // utilisation du v, ne ne crois pas qu'on puisse
set la valeur en faisant v = newLetter, alors je vais le garder comme ca, dans la vraie vie je l'aurais testé
rapido
       });
       return true; // dépendemment de l'utilisation, la fonction pourrait retourner void et donc ne pas avoir de
type de retour, à voir
   }
   return false; // suppression de code inutile
}
```

Voici les défauts qu'il fallait pointer dans le code (soit les corriger, soit les discuter dans le texte):

Expression booléenne (1 pt)
L'expression ternaire est complètement inutile et devrait être éliminée (0,5 pt). De plus, on n'a pas besoin de la variable correct (0,5). On doit appeler directement la méthode isValidItem().

Renommage des variables (1 pt)
Le paramètre "a" n'est assurément pas un bon nom (0,5 pt). Idem pour les paramètres "v" et "i" (0,5 pt)

Fonction obtenirAleatoire() (1 pt)
Elle devrait être en anglais comme les autres (0,5 pt). De plus, comme elle prend un paramètre, le nom de la fonction ne semble pas adéquat (0,5 pt). Soit on considère qu'il faut enlever le paramètre, soit on change son nom pour quelque chose comme exchangeLetter().

Attribut this.items (0,5 pt)
On voit pas très pourquoi il existe en parallèle avec this.letters, et en plus le nom n'est pas adéquat. Il semble que un des deux ne devrait pas exister. De plus, this.letters est un Map qu'on parcourt de manière non habituelle (par les valeurs plutôt que par les clés)

Logique du code (0,25)
On se demande pourquoi on change toutes les occurrences de la lettre par une même lettre. Ici, pas nécessaire de corriger l'erreur, mais on doit au moins pointer ce défaut potentiel. Pour corriger, soit on enlève la variable temporaire et on fait l'appel à chaque occurrence de la lettre trouvée, soit on modifie le code pour ne changer que la première occurrence de la lettre (mais on ne fait pas ça en mettant un break dans le forEach())

Pas de nouveau défaut introduit dans votre nouvelle version du code (0,25)

À noter qu'il y a d'autres défauts qui ont été exclus du barème de correction, mais qui pourraient amener à être moins sévère sur le reste si vous les avez identifiés, notamment:

- La méthode ne devrait pas retourner un booléen
- le nom isValidItem() n'est pas adéquat
- La validation de la lettre devrait être faite avant d'appeler la méthode

La méthode améliorée pourrait ressembler à ceci:

```
replaceLetter(letterToReplace: Letter): void {
   this.playerLetters[this.playerLetters.indexOf(letterToReplace)] = letterToReplace;
}
```

Commentaire :
renommage getRandom() -0,5

this.items -0,5

Question **11**

Partiellement correct

Note de 0,40 sur 1,00

Pourquoi faut-il toujours exécuter les tests après avoir amélioré un segment de code?

- a. Pour vérifier que les tests sont toujours de bonne qualité. ✖
- b. Pour s'assurer que les modifications n'ont pas créé de nouveaux défauts dans le code. ✖
- c. Pour s'assurer que le code est toujours fonctionnel après les changements. ✔
- d. Pour vérifier que les changements apportés sont de bonne qualité. ✖
- e. Pour s'assurer du succès des tests qui seront exécutés lors du *Merge Request*. ✖

Votre réponse est partiellement correcte.

Vous avez sélectionné trop d'options.
La réponse correcte est :
Pour s'assurer que le code est toujours fonctionnel après les changements.

Question **12**

Terminer

Note de 2,00 sur 2,00

Présentez deux (2) bonnes pratiques à suivre lorsque vous êtes responsables de faire la revue de code d'une demande fusion (*Merge Request*) sur GitLab.

C'est génial la revue de code directement sur GitLab !!!

D'abord c'est bien de la faire avant d'accepter la MR (même si c'est possible de le faire apres).

Ensuite en le faisant, c'est bien de selectionner le lignes concernees directement sur le UI de GitLab puis ajouter un commentaire en tagant @Nom l'auteur concerne.

L'auteur concerner doit lire les commentaire en dicuter (idealement directement la review de la MR).

Ensuite modifer les changements en consequences sur sont local, add, commit, push (qui trigger le pipeline automatiquement).

Il faut que la version avec les nouvelles modifications passe idealement le pipeline.

Evidemment, le reviewer ne doit pas etre l'auteur ni avoir aidé de trop près à coder la MR sinon il pourait subir le biais de l'auteur (ce que j'ai fait, c'est bien (hum, pas toujours !))

Tout ça ce sont de bonnes pratiques ^^

Commentaire :

◄ Annonces

Aller à...

Introduction ►