Trinomes: Abdullah NEZAMI, Thomas BARSEGHIAN & Jean GIESE
Group: TP5
Projet: SAÉ 11 & 12 - Jeu de Yams

Rapport SAE 11 + SAE 12

Partie P11

Structures et tableau :

3 Structures :

La structure Player sert à définir les 2 joueurs et stocker les informations nécessaires. Ils auront tout les 2 un identifiant, un pseudo, un score et un bonus

La structure ResultatsParRound contient l'identifiant du joueur, un tableau avec les dés qu'il a reçu, le nom du challenge qu'il a choisi et le score qu'il a obtenu sur ce round

Enfin la structure Rounds contient le numéro du tour et un tableau de structure ResultatsParRound

2 Tableaux :
On a aussi 2 tableaux de chaîne de caractères à 3 dimentions, challengesMin et challengesMaj qui servent à afficher les challenges possibles aux joueurs. Les challenges sont stockés 2 fois dessus, une fois pour chaque joueur. Ce tableau sera modifié lorsque des challenges seront enlevés du tableau.

Structure générale du programme (fonctions, procédures) :

initialiserPlayers :

Idée: Initialiser le joueur en lui demandant son nom et en lui attribuant un identifiant unique

Entrée: Player[] players: Tableux de structure player
Local: nom : chaine de caractere
Sortie: void

afficherTab :
Petite fonction qui sert à afficher un tableau d'entier sous forme d'entier

afficherTabDe :
Fonction qui sert à afficher un tableau d'entier sous forme de dés en utilisant un dictionnaire qui relie un entier à une Liste de chaine de caractères

lanceDe :
Fonction qui rempli un tableau reçu en paramètre par un entier entre 1 et taille de tableau + 2 (non inclus) donc lorsqu'on lance 5 dés, on aura 5 chiffres compris entre 1 et 7 non inclus donc au final 5 chiffres compris entre 1 et 6

relance :
Fonction qui prend le tableau d'entiers en paramètre (le tableau de dés) et remplace les dés que le joueur souhaite relancer par des 0 (facilite la tâche pour la fonction lanceEtFusionne)
De multiples try-catch sont utilisé pour gérer les erreurs de saisies

lanceEtFusionne :
Les 0 dans le tableau (=les dés qu'on souhaite relancer) sont remplacé par des chiffres entre 1 et 6

startGame :
La fonction coeur du programme, startGame initialise les variables clés et exécute les 13 rounds en gérant l'affichage en console des interfaces utilisateurs, des challenges et des dés. Cette fonction en appelle d'autres comme afficherChallenges, lanceDe, afficheTabDe, caseChecker et relance.

afficherChallenges :

Fonction qui affiche les challenges restant du joueur en vérifiant si le challenge a déjà été choisi et aussi en vérifiant si tous les challenges de ce type (mineur ou majeur) ont été choisi, si c'est le cas la fonction n'affiche pas les entêtes pour ce type

choixChallenge :
La fonction choixChallenge, comme son nom l'indique, s'occupe de toute la partie du choix du challenge. Elle récupère d'abord le numéro du challenge choisi par le joueur. La fonction vérifie si le numéro indiqué est en effet un challenge possible à choisir pour ce joueur et ensuite elle traite si le challenge choisi est un challenge mineur ou majeur. Elle appelle ensuite removeChallenge pour enlever le challenge choisi de la liste des challenges possibles pour ce joueur. La fonction crée ensuite une nouvelle structure ResultatsParRound pour stocker les informations. Et enfin si le challenge choisi est un challenge mineur, la fonction ajoute les points nécessaires au compteur de points des challenges mineurs pour calculer le bonus.

removeChallenge :
Appelle getChallengesRestants pour connaître le nombre de challenge restant du même type que celui qu'on souhaite effacer. On écrase ensuite le challenge qu'on souhaite enlever en déplaçant toutes les lignes par 1. On remplace ensuite la dernière ligne par une chaîne vide pour qu'elle ne soit pas affichée au prochain tour.

getChallengesRestants :
Prends en argument le numéro du joueur et un type de challenge et renvoie le nombre restant de challenges de ce type pour ce joueur

ScoreCalcule :
Appelle les fonctions nécessaires en fonction du challenge choisi pour vérifier si le challenge est validé. Si le challenge est validé, la fonction accorde les points gagnés au joueur. Il renvoie ensuite le score gagné par le joueur.

caseChecker :
Convertisseur de casse pour tout transformer en majuscule, utile pour corriger des erreurs de saisie, elle est utilisée lorsque l'utilisateur indique s'il veut relancer ses dés ou non. Dès que la fonction détecte une lettre en minuscule elle lui enlève 32 pour avoir l'équivalent en majuscule sur la table ASCII.

FichierJSON :

Comme nous avons déjà différentes structures et tableaux pour toutes les données, nous avons conçu cette fonction avec une boucle qui utilise toutes les structures et génère un fichier JSON à l'aide de la fonction StreamWriter, comme demandé dans le projet. Cette fonction remplit le fichier JSON avec les informations collectées durant la partie.

Main :
La fonction Main affiche certains éléments de l'interface utilisateur sur le terminal, elle initialise aussi des structures clés au programme. Elle appelle ensuite startGame pour lancer les 13 rounds. Une fois startGame fini la fonction Main donne le bonus aux joueurs qui ont rempli les contraintes et enfin il annonce le gagnant de la partie. Et à la toute fin la fonction appelle FichierJSON pour créer le fichier JSON.

Toutes les fonctions de vérifications de challenges (en algorithme) :

sommeMemeNombre :
Idée : verifie si les dés apparaissent le bon nombre de fois dans le tableau et si c'est le cas il distribue le nombre de points nécessaire

Entrée : tabDe : tableau d'entier (contient les dés du joueur)
Times : entier representant le nombre de fois qu'on souhaite retrouver
Local : number : le chiffre qui apparaît le nombre de fois nécessaire
I : compteur
Sortie : times * number (correspond au nombre de points à distribuer)

```
Fonction sommeMemeNombre(tabDe : tableau d'entier, times : entier) : entier
Debut
    entier number = 0
    Pour i allant de 0 à 3
        Si conterDe(tabDe, tabDe[i]) == times
            Number = tabDe[i]
        Fin Si
    Fin Pour
    renvoyer (times * number)
Fin
```

conterDe :

Idée : compte le nombre d'occurrence du dé souhaité dans le tableau de dé reçu en paramètre

```
Fonction compterDe(tabDe : tableau d'entier, num : entier)
Debut
        entier compteur= 0
        Pour i allant de 0 à 5
                Si tabDe[i] == num
                        compteur = compteur + 1
                Fin si
        Fin Pour
        Renvoyer compteur
Fin
```

triTab :

Idée : tri un tableau avec la méthode par insertion

```
Fonction triTab(triTab : tableau d'entier) : void
Debut
        Soit temp un entier (on defini la variable temp)
        Pour i allant de 1 à longueur de tri Tab
                Si triTab[ i - 1 ] > triTab[ i ]
                        Pour j allant de i à 0 (avec un pas de -1)
                                Si triTab[j-1] > triTab[ j ]
                                        temp = triTab[j -1]
                                        triTab[j - 1] = triTab[ j ]
                                        triTab[ j ] = temp
                                Fin Si
                        Fin Pour
                Fin Si
        Fin Pour
Fin
```

fullChallenge :

Idée : Vérifie si le tableau donnée en paramètre vérifie les requis pour le challenge full (= contient 3 fois un même élément et 2 fois un autre)

```
Fonction fullChallenge(tabDe : tableau d'entier) : booléen
```

```
Debut
        Entier triple = 0
        Entier doub = 0
        Entier temp = 0
        Booleen isTrue = false
        Pour i allant de 0 à 5
                temp = conterDe(tabDe, tabDe[i])
                si temp == 3
                        triple = 3
                Fin si
                Si temp = 2
                        doub = 2
                Fin si
        Fin Pour
        Si triple == 3 et doub == 2
                isTrue =true
        Fin si
        Renvoyer isTrue
Fin
```

petiteSuite :
Idée : utilise triTab pour trier le tableau par ordre croissant et ensuite regarde si le tableau des dés contient 4 chiffres qui se suivent

```
Fonction petiteSuite(tabDe : tableau d'entier) : booléen
Debut
        booléen isTrue = false
        triTab(tabDe)

        entier oneToFour = 1
        entier tempOneToFour = 0

        entier  twoToFive = 2
        entier tempTwoToFive = 0

        entier threeToSiTx = 3
        entier tempThreeToSix = 0
```

```
        Pour i allant de 0 à 5
        Si oneToFour == tabDe[i] et tempOneToFour < 4
                        tempOneToFour++
                        oneToFour++
                Fin si
        Si twoToFive == tabDe[i] et tempTwoToFive < 4
                        twoToFive++
                        tempTwoToFive++
                Fin si
                Si threeToSiTx == tabDe[i] && tempThreeToSix < 4
                        threeToSiTx++
                        tempThreeToSix++
                Fin si
        Fin Pour
        Si tempOneToFour == 4 ou tempTwoToFive == 4 ou tempThreeToSix == 4
        isTrue = true
        Fin si
        return isTrue
Fin
```

grandeSuite :

Idée : Fonction quasi identique à petiteSuite, elle utilise triTab pour trier le tableau par ordre croissant et ensuite regarde si le tableau des dés contient 5 chiffres qui se suivent

```
Fonction petiteSuite(tabDe : tableau d'entier) : booléen
Debut
        booléen isTrue = false
        triTab(tabDe)
        entier oneToFive = 1
        entier tempOneToFive = 0

        entier  twoToSix = 2
        entier tempTwoToSix = 0

        Pour i allant de 0 à 5
        Si oneToFive == tabDe[i] et tempOneToFive < 5
                        tempOneToFive++
                        oneToFiver++
                Fin si
        Si twoToSix == tabDe[i] et tempTwoToSix < 5
```

```
                twoToSix++
                tempTwoToSix++
            Fin si
        Fin Pour
        Si tempOneToFive == 5 ou tempTwoToSix == 5
        isTrue = true
        Fin si
        return isTrue
Fin
```

Partie W11

Introduction

Le projet présenté dans ce rapport consiste en la conception et le développement d'un site web interactif pour afficher les résultats du jeu Yam's à l'aide d'une API. L'objectif principal était de créer une interface utilisateur intuitive et réactive, capable de gérer et de présenter les données JSON extraites de l'API, tout en garantissant une expérience utilisateur optimale.

Structure générale des pages HTML

1. **Input utilisateur** : Cette section permet à l'utilisateur d'entrer un numéro de fichier pour valider la partie. Si le numéro est incorrect, une alerte est affichée et les sections suivantes restent cachées.
2. **Vue Globale** : Une fois que le numéro est validé, cette section affiche les résultats globaux, comprenant les scores et les bonus des joueurs.
3. **Vue par Tour** : Accessible via un bouton dans la barre de navigation, cette section permet de visualiser les détails de chaque tour, présentés sous forme de carrousel interactif. L'utilisateur peut naviguer entre les tours grâce à des boutons "Précédent" et "Suivant".

Mise en forme avec CSS et Flexbox

Pour la conception du site, on a utilisé le CSS afin de créer un design élégant, attrayant et facile à utiliser. J’ai principalement utilisé **Flexbox**, ce qui nous a permis de positionner les différents éléments de manière flexible et bien organisée sur le site.

Pour assurer la **responsivité**, nous avons employé des **media queries**, qui permettent d'adapter le site à tous les types d'appareils, qu'il s'agisse de mobiles, de tablettes ou d'ordinateurs. Une méthode que je trouve particulièrement utile est de définir une base pour les tailles en rem, ce qui simplifie les calculs et facilite l'adaptation des éléments dans le design responsive.

Méthodologie pour effectuer les requêtes vers l'API et insérer les données JSON dans la page HTML:

1. **Envoi de requêtes HTTP** :
    - Une fonction générique, fetchData(url), est utilisée pour envoyer des requêtes HTTP à l'API. Cette fonction utilise fetch() pour récupérer les données en format JSON.
    - En cas d'erreur (comme une URL invalide ou une connexion échouée), la fonction gère l'exception en affichant un message d'alerte et en masquant les sections de résultat.
2. **Traitement des données JSON** :
    - Les données JSON récupérées sont passées à des fonctions spécifiques pour extraire les différentes informations nécessaires. Par exemple :
        - ParametersApi() récupère les détails de la partie (date et groupe).
        - ResultsApi() collecte les noms et ID des joueurs.
        - scoreBonusApi() affiche les scores finaux et les bonus.
3. **Insertion dynamique dans le DOM** :
    - Les éléments HTML correspondants sont sélectionnés via document.querySelector() ou document.querySelectorAll().
    - Les valeurs extraites des données JSON sont insérées dans le contenu HTML en mettant à jour les propriétés innerHTML ou textContent.
    - Par exemple, les scores des joueurs sont insérés dynamiquement dans les éléments <p> de la section "Vue Globale".
4. **Mise à jour conditionnelle** :
    - Des vérifications sont effectuées pour gérer l'affichage dynamique. Par exemple, dans scoreBonusApi(), une condition détermine quel joueur est gagnant et met à jour l'affichage en conséquence.
5. **Navigation dynamique** :

- Pour la "Vue par Tour", une logique de carrousel est mise en place. Chaque clic sur les boutons de navigation appelle une requête pour charger les données du tour correspondant, qui sont ensuite insérées dans la page.

Conclusion

Ce projet nous a permis de comprendre et de maîtriser plusieurs aspects du développement web, notamment :

1. **Gestion des requêtes API et gestion des erreurs** : Nous avons appris à implémenter des requêtes robustes, à gérer les échecs de connexion et à présenter des messages d'erreur clairs aux utilisateurs.
2. **Responsivité** : Une grande leçon que nous avons apprise est de toujours commencer la conception responsive par le design mobile, puis de s'étendre progressivement aux écrans plus larges. Cette approche simplifie le code et offre de meilleurs résultats.
3. **Limitation rencontrée** : Nous avons essayé de créer deux pages distinctes pour "Vue par Tour" et "Vue Globale", mais nous avons rencontré un problème. Chaque changement de page obligeait l'utilisateur à entrer de nouveau le numéro de fichier. Étant donné que nous n'avons pas étudié comment construire une application monopage (SPA), je nous n’avons pas réussi à maintenir les données entrées persistantes entre les pages.

Globalement

Taux de réussite estimé : 100%

(Forcément on peut toujours rajouter de nouvelles fonctionnalités mais en tout cas tout nous avons fait tout ce qu’on devait et voulait faire)

Ressenti de la SAE : le sujet était fun et permettait toujours de rajouter de nouvelles fonctionnalités et d'améliorer les anciennes. Et le niveau exigé était cohérent avec le niveau des TD et TP.