ChronoSpacer - Part II

Ce TP peut être réalisé au choix dans le projet ChronoSpacer commencé lors du tutoriel SFML ou sur un nouveau projet console.

Le projet ChronoSpacer va être complété avec les règles suivantes :

Bataille :

- Le joueur va affronter plusieurs adversaires au cours de « batailles »,
- Une bataille est une succession de plusieurs « manches ». Une manche est un jeu de timing qui consiste à appuyer au bon moment sur la barre espace,
- Chaque manche peut être gagnée ou perdue,
- La première manche commence automatiquement lors du début de la bataille,
- Dès qu'une manche est terminée, la manche suivante commence,
- La bataille prend fin lorsque toutes les manches ont été jouées.

Manche :

- Au début d'une manche, un message de départ est affiché. Ce message indique au joueur une durée « cible » au bout de laquelle il doit appuyer sur la barre espace.
- Lorsque le joueur appuie sur la barre espace, la manche en cours prend fin.
  - La durée en seconde depuis le début de la manche est calculée.
  - L'écart entre la durée de la manche et la durée cible est calculé. Si cet écart est inférieur à +/- une valeur de difficulté fixe, alors la manche est gagnée, sinon la manche est perdue.
  - Les informations de la manche sont affichées au joueur : durées, écart, résultat.

I – Manche :

Créer les fichiers Round.cpp et Round.h. Ces fichiers contiendront le code relatif à la gestion d'une manche (round).

Dans Round.h, ajouter la commande préprocesseur suivante en en-tête :

```
#pragma once
```

Dans Round.h, créer la structure "struct round" avec le type synonyme (typedef) Round. La structure contient les informations suivantes :

- targetDuration : La durée cible (en secondes)
- difficulty : La difficulté de la manche (en secondes)
- startTime : Le temps de départ de la manche (en secondes). -1 si la manche n'est pas commencée

- endTime : La temps de fin de la manche (en secondes). -1 si la manche n'est pas terminée
- isVictory : Le résultat de la manche (vrai ou faux)

Dans Round.cpp, inclure le fichier Round.h.
Implémenter les fonctions suivantes dans Round.cpp. Pour chaque fonction implémentée, ajouter sa déclaration dans Round.h :

- CreateRound : Initialise une valeur de type Round.
  - Paramètres : Durée cible, difficulté.
  - Valeur de retour : Manche initialisée.
  - Note : Penser à initialiser chaque variable membre du type Round à une valeur adéquate.

- StartRound : Débute une manche donnée. Affiche les informations de début de manche.
  - Paramètres : Manche à débuter, temps de début
  - Valeur de retour : Manche modifiée

- EndRound : Termine une manche donnée. Calcule le résultat et affiche les informations de fin de manche.
  - Paramètres : Manche à terminer, temps de fin
  - Valeur de retour : Manche modifiée

Inclure le fichier Round.h dans le fichier source principal du projet ChronoSpacer.cpp pour tester l'implémentation des fonctions.

II – Bataille :

Créer les fichiers Battle.cpp et Battle.h. Ces fichiers contiendront le code relatif à la gestion d'une bataille (battle).

Répéter les opérations relatives à l'inclusion d'un fichier d'en-tête pour Battle.h et Battle.cpp (#include et #pragma once).

En en-tête de Battle.h, déclarer l'existence du type Round :

```
typedef Round;
```

Inclure Round.h dans Battle.cpp.

En en-tête de Battle.h, déclarer une variable "externe" ROUND_MAX :

```
extern const int ROUND_MAX;
```

En en-tête de Battle.cpp, définir la variable ROUND_MAX.

```
const int ROUND_MAX = 10;
```

Dans Battle.h, créer la structure "struct battle" avec le type synonyme Battle. La structure contient les informations suivantes :

- rounds : Un tableau de 10 Round (utiliser la variable constante ROUND_MAX),
- roundCount : Le nombre de manches paramétrées de la bataille,
- currentRoundId : L'index de la manche actuelle,

Implémenter les fonctions suivantes dans Battle.cpp :

- CreateBattle : Initialise une valeur de type Battle.
  - Paramètres : /
  - Valeur de retour : La bataille initialisée.
  - Note : Penser à initialiser chaque variable membre de la structure Battle à une valeur adéquate. Pour l'instant, on ne se soucie pas de l'initialisation des manches.

- AddRoundToBattle : Ajoute une manche à la bataille.
  - Paramètre : Manche à ajouter, bataille à modifier,
  - Valeur de retour : Bataille modifiée.

- _StartRound : Démarre la manche actuelle d'une bataille.
  - Paramètres : Bataille à modifier, temps de début de round
  - Valeur de retour : Bataille modifiée.
  - Note : Ne pas ajouter cette fonction au fichier d'en-tête Battle.h. Cette fonction est interne à Battle.cpp et ne sera jamais utilisé par un autre cpp.

- StartBattle : Démarre la bataille en paramètre. Utilise _StartRound.
  - Paramètres : Bataille à modifier, temps de début de la bataille
  - Valeur de retour : Bataille modifiée.

- OnBattleReceiveAction : Applique l'effet de la barre espace sur la bataille en paramètre. Termine la manche en cours et joue la manche suivante. Utilise _StartRound.
  - Paramètres : Bataille à modifier, temps d'enregistrement de l'action
  - Valeur de retour : Bataille modifiée.
  - Note : Bien vérifier l'avancement de la bataille avant de changer de manche.

- IsBattleFinished : Indique si une bataille est terminée.
  - Paramètres : Bataille à évaluer.
  - Valeur de retour : Vrai ou faux.

- GetBattleVictoryCount : Compte le nombre de manche gagnées d'une bataille donnée.
  - Paramètre : Bataille à évaluer.
  - Valeur de retour : Nombre de manches gagnées.

Inclure le fichier Battle.h dans le fichier source principal du projet ChronoSpacer.cpp pour tester l'implémentation des fonctions.

III – Copies de valeurs :

Pour réaliser un test complet sur une Bataille avec trois manches, on appelle :

- 1 x CreateBattle
- 3 x AddRoundToBattle
- 1 x StartBattle
- 3 x OnBattleReceiveAction
- 3 x IsBattleFinished
- 1 x GetBattleVictoryCount

Estimer approximativement :

- Le nombre de variables de type Battle allouées en mémoire,
- Le nombre de valeurs de type Battle copiées,
- Le nombre de variables de type Round allouées en mémoire,
- Le nombre de valeurs de type Round copiées.

Expliquer le résultat et les différentes modifications pouvant rendre le code plus optimisé.

IV – Références :

Dans toutes les fonctions de Battle.cpp, à l'exception de CreateBattle :

- Remplacer les paramètres de fonction de type Battle par des paramètres de type référence vers Battle.
- Retirer les valeurs de retour de type Battle.

Adapter le code de ChronoSpacerUtils.cpp à ces modifications.

Estimer approximativement :

- Le nombre de variables de type Battle allouées en mémoire,
- Le nombre de valeurs de type Battle copiées,
- Le nombre de variables de type Round allouées en mémoire,
- Le nombre de valeurs de type Round copiées.

V – Aléatoire :

Pour faire varier la durée cible d'un round à l'autre, on souhaite tirer ce temps de manière aléatoire. Le temps tiré aléatoirement sera une valeur entière en seconde.

Créer les fichiers ChronoSpacerUtils.h et ChronoSpacerUtils.cpp. Ces fichiers contiendront des données et fonctionnalités utilitaires communes à l'ensemble du projet.

Répéter les opérations relatives à l'inclusion d'un fichier d'en-tête pour ChronoSpacerUtils.h et ChronoSpacerUtils.cpp (#include et #pragma once).

Ecrire la fonction suivante dans ChronoSpacerUtils.cpp :

- GetRandomInt : Renvoie une valeur entière comprise dans un intervalle de valeurs.
  - Paramètres : Borne minimale entière inclue, borne maximale entière exclue.
  - Valeur de retour : Valeur aléatoire entière.

- InitRandom : Initialise la graine du tirage random pour l'ensemble du programme
  - Paramètres : /
  - Valeur de retour : /

Pour écrire la méthode, inclure cstdlib et ctime au fichier ChronoSpacerUtils.cpp. On utilisera alors les fonctions suivantes pour le tirage aléatoire :

```
srand(time(NULL)); // Random seed initialization (must be called once)
rand(); // Random int between 0 and RAND_MAX (excluded)
```

Utiliser l'opérateur modulo pour réaliser le tirage aléatoire entre deux bornes :

```
int result = min + rand() % (max - min);
```

Inclure le fichier ChronoSpacerUtils.h dans le fichier source principal du projet ChronoSpacer.cpp pour tester l'implémentation des fonctions.

VI – Boucle de jeu

Dans ChronoSpacer.cpp, écrire une boucle de jeu (while) permettant de jouer une bataille entière. La boucle de jeu s'arrête lorsque IsBattleFinished est vrai.

Avec SFML :
- Traiter les inputs claviers à chaque boucle de jeu. Appeler OnBattleReceiveAction lorsque le joueur appuie sur la barre espace.
- Utiliser sf::clock pour calculer le temps depuis le lancement du programme.

Sans SFML :
- Mettre le jeu en pause à chaque tour de boucle avec system("pause"). Après la reprise d'exécution, appeler OnBattleReceiveAction.
- Utiliser ((float)clock() / CLOCKS_PER_SEC) grace à la bibliothèque ctime pour calculer le temps écoulé depuis le lancement du programme.

VII – Extra : Timeout

Avec SFML, intégrer au système de manche une fonctionnalité de timeout. Si au bout de plusieurs secondes, le joueur n'a pas appuyé sur la barre espace, alors le round est automatiquement perdu.

Pour réaliser la fonctionnalité de timeout. Ajouter une fonction UpdateBattle dans Battle.cpp et une fonction IsRoundTimedOut dans Round.cpp. Appeler la fonction UpdateBattle à chaque boucle de jeu.

VII – Extra : Display

Avec SFML, écrire les fonctions DisplayBattle et DisplayRound respectivement dans Battle.cpp et Round.cpp.

- DisplayBattle : Affiche les informations relatives à une bataille à l'écran.
  - Paramètre : Référence vers la fenêtre SFML, Position d'encrage
  - Valeur de retour : /
  - Note : Ajouter les formes utilisées dans la structure de Battle. Initialiser les formes dans la fonction CreateBattle.

- DisplayRound : Affiche les informations relatives à un round à l'écran.
  - Paramètre : Référence vers la fenêtre SFML, Position d'encrage
  - Valeur de retour : /
  - Note : Ajouter les formes utilisées dans la structure de Round. Initialiser les formes dans la fonction CreateRound. Appeler DisplayRound depuis DisplayBattle.

Appeler DisplayBattle à chaque boucle de jeu SFML depuis ChronoSpacer.cpp.