Introduction à l'architecture d'un moteur de jeu vidéo

TP 2: La boucle de jeu

Abdelkader Gouaïch

2023/2024

Configuration de l'environnement de travail

Voici quelques rappels et recommandations pour configurer votre environnement de travail:

1. Utilisez Visual Studio Code comme votre environnement de développement intégré (IDE).
2. Installez les extensions nécessaires pour le développement Web et JavaScript afin de faciliter votre travail.
3. Ajoutez l'extension "Live Server" pour bénéficier d'un serveur de développement en temps réel.

En ce qui concerne la suite du TP, voici les étapes à suivre avant de commencer:

- Téléchargez le fichier "session2.zip" et décompressez-le. Ce fichier contient un projet préconfiguré que vous allez utiliser.
- Vous allez principalement travailler sur le fichier "engine/loop.js." Assurez-vous de ne modifier que ce fichier.
- Vous êtes libre de consulter les autres fichiers du projet, mais veillez à ne pas les modifier, car cela pourrait perturber la configuration préétablie.

Présentation du sujet

En premier lieu, nous allons examiner le contenu du fichier "src/myGame.js." Ce fichier contient une classe JavaScript qui définit une scène graphique. La scène graphique est initialisée dans la méthode "init()" de cette classe. C'est une scène simple qui se compose de deux carrés, l'un blanc et l'autre rouge.

Pour visualiser notre scène, nous avons également défini une caméra. Cette caméra est responsable du rendu de la scène sous forme d'image. L'image est un résultat calculée à partir des états des objets graphiques.

Pour simplifier, nous pouvons définir l'état d'un objet graphique comme l'ensemble des points qui composent sa géométrie, sa forme (comment les segments et les faces sont retrouvés à partir des points) et ses couleurs.

Nous avons défini deux primitives importantes.

La première, "draw()," est utilisée pour dessiner effectivement la scène.

La seconde, "update()," sert à mettre à jour les états de nos objets graphiques.

Comme vu en cours, définir une boucle de jeu revient à définir une stratégie d'ordonnancement de ces deux primitives.

Objectif

La fonction "update()" réalise plusieurs animations :

- Une rotation en appliquant une transformation de type rotation sur le carré blanc.
- Une animation de type mouvement en effectuant une translation le long de l'axe X sur le carré blanc.
- Une animation de type zoom (mise à l'échelle) en réalisant une homothétie sur le carré rouge.

Votre objectif est d'utiliser correctement les deux primitives "draw()" et "update()" depuis le fichier "loop.js" afin d'obtenir des animations fluides.

De plus, vous devez calculer le nombre d'images par seconde (FPS) et le nombre de mises à jour par seconde (UPS) pour chaque stratégie d'animation.

Etape 1: Boucle de rendu graphique

Dans cette étape nous allons réaliser uniquement le rendu de la scène *sans* mettre mettre à jour les états et par conséquence la scène sera figé sans aucune animation.

Instructions

Le module "loop.js" est pour l'instant très simple: il exporte deux fonctions exportées : `start` et `stop`. Ces fonctions seront utilisées par le module principal pour lancer le jeu et l'arrêter. Le module principale sera responsable de fournir un objet de type `MyGame` à la fonction `start`.

1. **Ajout de Constantes et Variables**:
   - Ajoutez les constantes globales `iUPS` (Updates Per Second) et `tMPF` (Milliseconds Per Frame) pour gérer le timing du jeu. `iUPS` donne le nombre de cyle par seconde souhaité pour les mises à jour et `tMPF` donne la durée d'un cycle en fonction de la fréquence souhaitée.
   - Déclarez les variables `tPrevTime`, `tLagTime`, `bLoopRunning`, `oCurrentScene`, et `iFrameID` pour stocker l'état du jeu et les informations de timing.

```
const iUPS = 60; // nombre de updates par seconde
const tMPF = 1000 / iUPS; // Milliseconds par cycle.
// Variables for timing gameloop.
let tPrevTime; // le dernier temps enregistré
let tLagTime; // lag: différence entre le dernier temps et maintenant
let bLoopRunning = false; // Etat pour savoir si la boucle est active.
let oCurrentScene = null; // reference vers la scène /jeu courant
let iFrameID = -1; //identifiant pour la requete retourne par requestAnimationFrame
```

2. **Développement des Fonctions de Boucle**:
   - Créez la fonction `loopOnce` qui sert de boucle principale du jeu exécutée pendant *une seule* itération.
   - Consultez la documentation de la fonction JS `performance.now()` qui sera utile pour mesurer le temps et mettre à jour les variables `tPrevTime` et `tLagTime` dans la fonction `loopOnce`
   - Développez une fonction supplémentaire `loopRender` qui gère le rendu graphique de la scène.

```
function loopOnce() {
  /* TODO */
}

function loopRender() {
  /* TODO */
}
```

3. **Amélioration des Fonctions `start` et `stop`**:
   - Consultez la documentation de la fonction JS `requestAnimationFrame()`
   - Consultez la documentation de la fonction JS `performance.now()`
   - Dans la fonction `start`, ajoutez du code pour initialiser l'état du jeu et démarrer la boucle de jeu.
   - Dans la fonction `stop`, ajoutez du code pour arrêter la boucle de jeu et annuler la prochaine animation frame.

```
// fonction start scene
function start(scene) {
  /* ... */
}
```

```
//arret de la boucle de jeu
function stop() {
  /* ... */
}
```

Une fois terminée, votre navigateur devrait afficher la scène suivante:

Figure 1: Résultat de l'étape 1

Etape 2: Boucle simple de render/update

Nous allons maintenant gérer les mises à jour des états des objets de notre scène.

Notre scène est composée de deux carrés qui sont animés:

- Un carré effectue une translation et une rotation
- L'autre carré change sa taille

Notre objectif est de prendre en compte ces animations qui sont *déjà* programmées dans la classe `MyGame`. Nous souhaitons donc appeler la fonction `MyGame.update` depuis notre boucle de jeu pour mettre à jour les états des ob-

jets. Notre stratégie d'appel des fonctions `MyGame.update` et `MyGame.draw` va influencer les métriques FPS et UPS.

Instructions

Vous devez partir du code de l'étape précédente.

1. **Développement des Fonctions de Boucle**:
   - Développez une fonction supplémentaire `loopUpdate` qui gère la mise à jour la scène en utilisant la fonction `update` de l'objet `MyGame`.
   - La stratégie de `loopUpdate` est très simple: elle met à jour la scène sans prendre en compte un éventuel lag
   - Modifier la fonction `loopOnce` pour appeler `loopRender` et `loopUpdate`.

```
function loopOnce() {
  /* TODO : modifier le code etape 1 */
}

function loopRender() {
  /* Code etape 1 */
}
function loopUpdate() {
  /* TODO */
}
```

Une fois cette étape terminée, vérifier que la scène sur le navigateur est animée.

Etape 2: Boucle améliorée de render/update

A présent, nous allons prendre en compte un éventuel retard ou lag pour la mise à jour de l'état de la scène.

Un lag se produit quand les événements de déclenchement du cycle de la boucle de jeu accumulent des retards. En effet, c'est `requestAnimationFrame` qui décide quand la fonction est `loopOnce` est effectivement appelée. La fréquence théorique des appels est synchronisée sur la fréquence de rafraîchissement de l'écran 60Hz; mais des retards sont possibles.

Nous proposons dans cette étape d'améliorer la fonction `loopUpdate` pour que, en cas de retard accumulé, nous allons itérer autant de fois jusqu'à rattraper notre retard.

Instructions

Vous devez partir du code de l'étape précédente.

1. **Modification des Fonctions de Boucle**:
    - Modifier la fonction `loopUpdate` pour qu'en cas de lag, nous itérons et appelons la mise à jour de la scène jusqu'à consommation de notre retard. Le temps de retard est accumulé dans la variable `tLagTime` et un cycle d'update théorique dure `tMPF`. Nous allons donc itérer tant que `tLagTime` est supérieur à `tMPF` en retranchons `tMPF` à chaque itération. Autrement dit, nous allons réaliser une division euclidienne de `tLagTime` sur `tMPF` pour déterminer le nombre d'itérations à réaliser.

```
function loopUpdate() {
  /* A modifier */
}
```

Une fois cette étape terminée, vérifier que la scène sur le navigateur est toujours animée.