IFT1015 – Travail pratique #2 – 2020-11-22

JEU DE ADDICTION SOLITAIRE

Marc Feeley

Le TP2 a pour but de vous faire pratiquer les concepts suivants : les boucles, les tableaux, les fonctions, la décomposition fonctionnelle, le traitement d'événements et la programmation web.

Le code que vous devez écrire (fichier "tp2.py") implante un jeu qui exécute dans l'environnement du navigateur web. Une bonne partie du jeu peut se développer avec codeBoot. Cependant, il faut comprendre qu'on vise à déployer le programme comme une application web standard (les détails sont expliqués ci-dessous).

Vous pouvez utiliser le code qui a été montré dans le cours mais vous ne devez pas utiliser du code provenant d'ailleurs, que ce soit du web ou d'une autre personne.

---

1 Introduction

Ce travail pratique consiste à développer un programme web pour jouer au jeu "addiction solitaire". C'est un jeu joué en solitaire (un seul joueur) avec des cartes à jouer standard (52 cartes).

Le jeu consiste à disposer les cartes brassées sur une grille de quatre rangées et treize colonnes, puis de retirer les as, ce qui crée quatre "trous" dans les rangées. Ensuite on doit déplacer les cartes une à la fois pour remplir les trous avec l'objectif d'avoir sur chaque rangée les cartes du 2 au roi d'une même couleur en séquence. On peut déplacer un 2 seulement sur un trou dans la première colonne. On peut déplacer les cartes du 3 au roi seulement dans un trou qui est à la droite de la carte de même couleur qui vient juste avant (par exemple, déplacer le valet de coeur dans un trou qui est à la droite du dix de coeur). Lorsqu'on déplace une carte, cela crée un nouveau trou, ce qui peut permettre un nouveau déplacement (ou plusieurs déplacements si on vient de créer un trou dans la première colonne).

À tout moment, et particulièrement lorsqu'il n'y a plus de déplacements de cartes possibles, on peut retirer du jeu les cartes qui ne sont pas encore dans la bonne position, les combiner avec les quatre as, les brasser, les remettre sur le jeu, et enlever les as pour créer à nouveau quatre trous. On peut le faire au maximum trois fois.

Le joueur gagne s'il place toutes les cartes en séquence sur les quatre rangées. Le joueur perd s'il ne peut plus déplacer de cartes et il a déjà brassé les cartes trois fois.

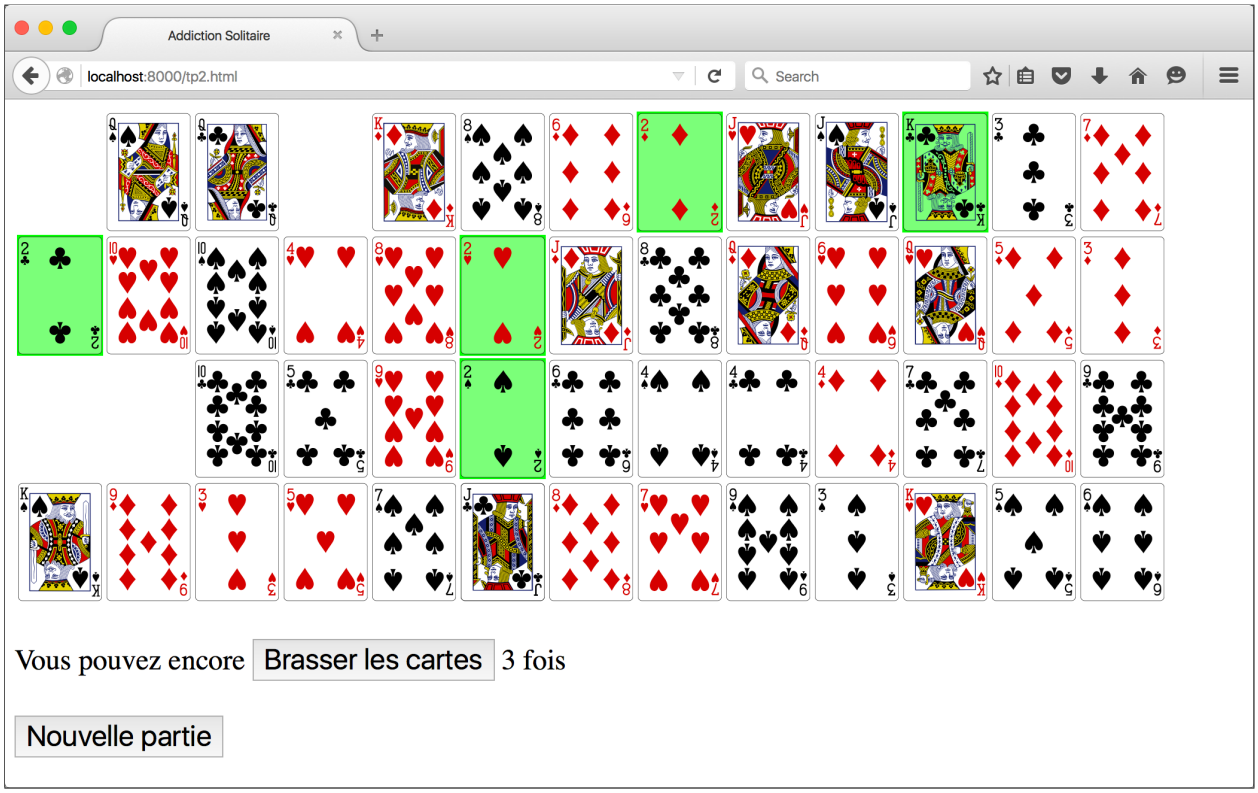
Un modèle complètement fonctionnel du jeu est disponible sur Studium, tel qu'expliqué ci-dessous. Votre travail consiste donc à imiter son comportement le plus fidèlement possible. Notez cependant que le modèle débute la première partie toujours avec le même agencement de cartes. Cela est purement pour vous aider à expérimenter avec le jeu pour bien comprendre son fonctionnement. Votre programme doit utiliser `random()` pour brasser les cartes.

2 Déroulement de la partie

L'interface graphique du jeu contient une grille de 4 rangées et 13 colonnes avec les images des cartes et les quatre trous. En dessous se trouve une ligne qui affiche un message et en dessous du message, un bouton

pour redémarrer une nouvelle partie. Pendant la partie, le message indique si on peut brasser les cartes, et si c'est le cas un bouton dans le message permet de le faire. Lorsque la partie est terminée, le message indique si le joueur a réussi ou non.

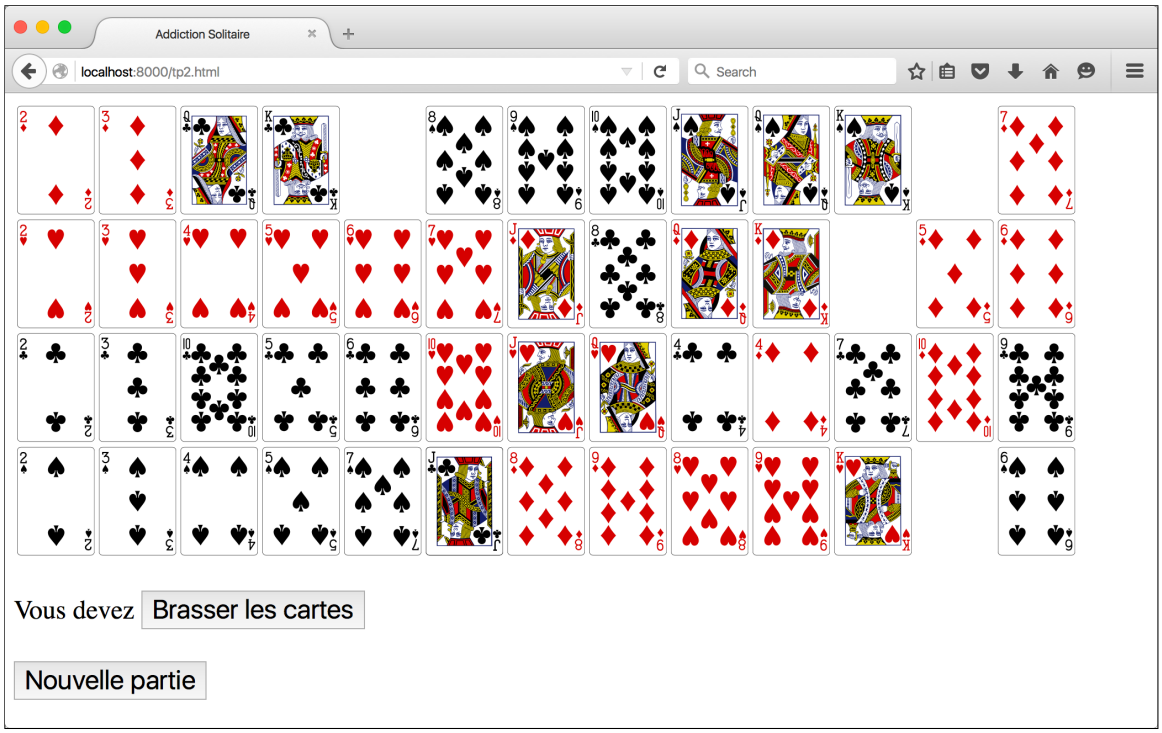
Voici à quoi ressemble la fenêtre au début d'une partie :



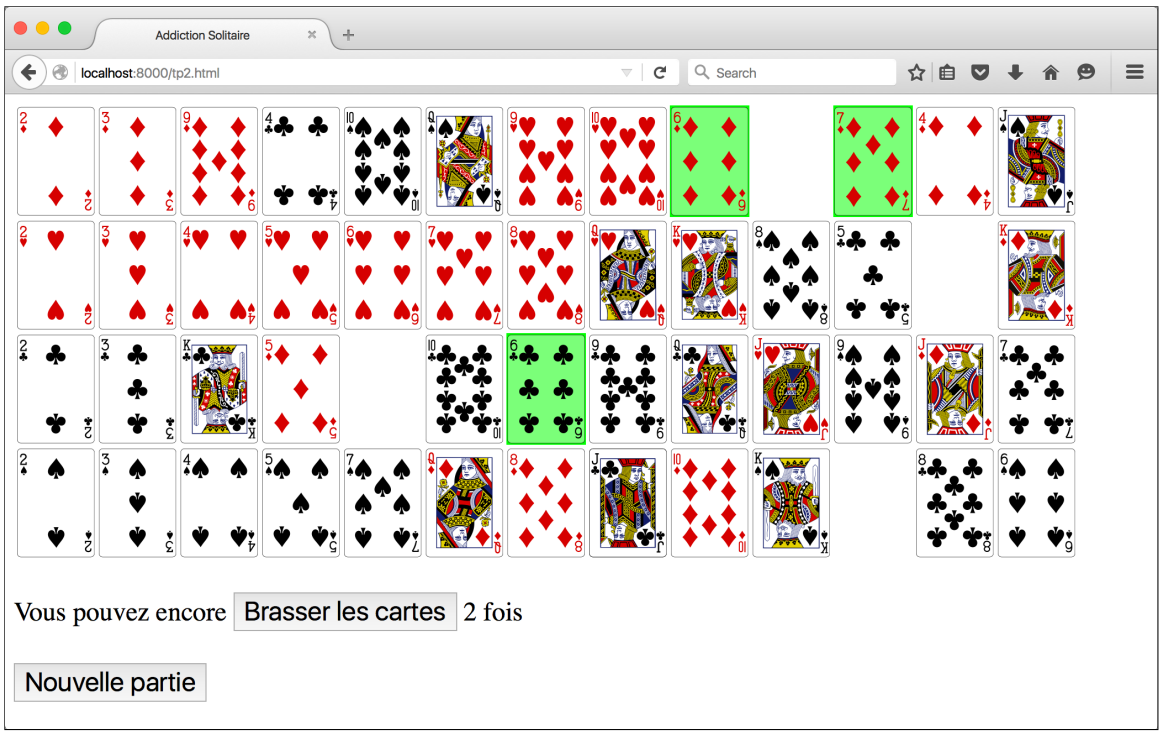
Pour aider le joueur, les cartes qui peuvent être déplacées ont un fond de couleur vert vif. Dans l'exemple ci-dessus, on peut déplacer le roi de treffle car il y a un trou à la droite de la dame de treffle. D'autre part, puisqu'il y a (au moins) un trou dans la première colonne, tous les 2 peuvent être déplacés, même le 2 qui est déjà dans la première colonne car on pourrait vouloir le déplacer ailleurs sur la première colonne.

C'est en cliquant sur une carte que le joueur la déplace. Souvent il y a seulement une destination possible, par exemple le roi de treffle. Pour les 2 il peut y avoir plus d'une destination possible. Dans ce cas, si le 2 n'est pas dans la première colonne, le jeu déplacera la carte au premier trou de la première colonne. Si le 2 est déjà dans la première colonne, le jeu déplacera la carte au prochain trou de la première colonne.

Voici à quoi pourrait ressembler la fenêtre lorsque le joueur a fait plusieurs déplacements et il ne reste plus de déplacements possibles :



Le joueur doit donc cliquer sur le bouton pour brasser les cartes. La fenêtre deviendra :



3 Détails techniques

Pour vous démarrer dans la bonne direction, vous trouverez sur Studium le fichier “serveur-web.py” et le fichier “documents.zip”. Le fichier “documents.zip” lorsque “dézippé” sur votre ordinateur va donner un répertoire “documents” qui peut être utilisé tel quel par le serveur “serveur-web.py” afin d'exécuter le programme modèle. En effet le répertoire “documents” contient les fichiers “modele.html”, “modele.css”, “modele.js” et “cards” (un sous-répertoire qui contient les images des cartes en format SVG, “Scalable Vector Graphics”). Donc si vous démarrez le serveur web avec la commande `python3 serveur-web.py` vous pouvez démarrer le modèle en utilisant votre navigateur web pour visiter l'URL `http://localhost:8000/modele.html` .

Les fichiers “tp2.html”, “tp2.py”, “codeboot.bundle.js” et “codeboot.bundle.css” sont également présents dans le répertoire “documents”. Vous ne devez pas changer cette organisation de fichier ni changer le contenu des fichiers sauf **“tp2.py” que vous devez compléter avec votre code.** Pour démarrer l'exécution de votre code visitez l'URL `http://localhost:8000/tp2.html` .

Le fichier “tp2.html” contient des directives pour inclure les fichiers “codeboot.bundle.js”, “codeboot.bundle.css” et le code Python “tp2.py”. Le corps du document est le suivant :

```
<body onload="init()">
  <div id="main"></div>
</body>
```

Le traiteur d'événement `onload` du corps fait donc un appel à la fonction `init` définie dans “tp2.py” pour démarrer l'exécution du code Python au moment du chargement du fichier “tp2.html”. La fonction `init` doit créer le contenu HTML qui sera mis dans l'élément `<div id="main"></div>`. On pourrait par exemple définir la fonction `init` comme suit pour afficher une grille 2x2 de cartes avec une des cartes sélectionnée :

```
def init():
    main = document.querySelector("#main")
    main.innerHTML = """
      <style>
        #jeu table { float: none; }
        #jeu table td { border: 0; padding: 1px 2px; height: auto; }
        #jeu table td img { height: auto; }
      </style>
      <div id="jeu">
        <table>
          <tr>
            <td id="case0"><img src="cards/2S.svg"></td>
            <td id="case1"><img src="cards/QH.svg"></td>
          </tr>
          <tr>
            <td id="case2"><img src="cards/JC.svg"></td>
            <td id="case3"><img src="cards/10D.svg"></td>
          </tr>
        </table>
      </div>"""

    case0 = document.querySelector("#case0")
    case0.setAttribute("style", "background-color: lime")
```

Évidemment, pour une grille de 13x4 il serait mieux de créer le HTML avec des boucles pour éviter des répétitions de code.

Les noms des fichiers d'image de cartes dans le sous-répertoire "cards" sont dérivés de l'anglais ("QH.svg" pour "Queen of Hearts" et "10D.svg" pour "10 of Diamonds"). Le format SVG a l'attrait de donner des images très nettes car c'est un format qui permet de faire un zoom infini sur l'image.

Pour indiquer qu'une carte est sélectionnée, utilisez la couleur `lime` comme `background-color` du style de l'élément. Pour désélectionner une carte retirez l'attribut de style de l'élément (avec la méthode `removeAttribute`).

Pour représenter les cartes dans votre programme il est suggéré d'utiliser un nombre de 0 à 51 pour identifier les cartes. Les nombres de 0 à 3 correspondent aux 4 as (de chaque couleur), puis les nombres 4 à 7 correspondent aux 4 "2" (de chaque couleur), et ainsi de suite. Donc si on a les cartes x et y, elles ont la même couleur si $x\%4 = y\%4$, et elles ont la même valeur si $x//4 = y//4$.

Un paquet de carte peut donc être représenté par un tableau de longueur 52 contenant des nombres de 0 à 51 (dans un ordre arbitraire si le paquet est mélangé). On peut créer un paquet de carte mélangé en partant d'un tableau des nombres consécutifs de 0 à 51, puis échanger le dernier élément avec un des éléments du tableau pigé aléatoirement, puis échanger l'avant dernier élément avec un des éléments du tableau (sauf le dernier) pigé aléatoirement, puis échanger l'avant avant dernier élément avec un des éléments du tableau (sauf les deux derniers) pigé aléatoirement, et ainsi de suite.

Notez que le développement du programme peut se faire de deux façons. Vous pouvez éditer directement le fichier "tp2.py" avec un éditeur de code et utiliser le navigateur pour raffraichir la page `http://localhost:8000/tp2.html` (ce qui va exécuter votre programme "tp2.py" à nouveau). Un clic avec le bouton de droite vous permet d'afficher dans une fenêtre flottante l'état du programme, ce qui est utile s'il y a bogues ou des appels à `breakpoint()` dans votre code.

Vous pouvez aussi développer votre programme "tp2.py" directement dans codeBoot. Pour démarrer l'exécution de codeBoot visitez l'URL `http://localhost:8000` . Dans votre programme, vous pouvez soit utiliser `main = document.querySelector("#main")` comme l'exemple ci-dessus (pour afficher le jeu au dessus de codeBoot) ou faire `main = document.querySelector(".cb-html-window")` (pour afficher le jeu au même endroit que la fenêtre de pixels). Avant la remise, n'oubliez pas de tester votre code au moyen du serveur web pour vous assurer que tout est fonctionnel (entre autre il faut utiliser `#main` dans la version que vous remettez).

4 Évaluation

- Ce travail compte pour 12.5 points dans la note finale du cours. Vous devez le faire par groupes de 2 personnes. Indiquez vos noms clairement dans les commentaires au début de votre code.
- Vous devez remettre votre fichier "`tp2.py`" **uniquement**. La remise doit se faire au plus tard à midi lundi le 14 décembre sur le site Studium du cours.
- Chaque fonction devrait avoir un bref commentaire pour dire ce qu'elle fait, il devrait y avoir des lignes blanches pour que le code ne soit pas trop dense, les identificateurs doivent être bien choisis pour être compréhensibles et respecter le standard CamelCase.