Le jeu est constitué de plusieurs parties



**Règles du jeu**

Le jeu démarre automatiquement au chargement.

Le joueur et l'ordinateur commenceront chacun avec 7 cartes, et une carte numérique commencera la pile de jeu. Le joueur commence. Il peut soit cliquer sur une carte de valeur ou de couleur correspondante pour la jouer, soit jouer une carte d'action (Inverser, Sauter, Tirer 2, Tirer 4, Joker), soit, si aucune carte jouable n'est disponible, cliquer sur la pile de tirage pour obtenir une nouvelle carte et renoncer à son tour.

L’ordi joue ensuite, soit en jouant une carte appropriée, soit en prenant une carte dans le talon.

Les cartes Draw 2 (+2) et Draw 4 (+4) ajoutent automatiquement leur montant à la main de la victime et font avancer le tour. Les cartes Reverse et Skip sautent le tour de la victime (comme il n'y a que deux joueurs, Reverse devient essentiellement un Skip). Les cartes +2 ou +4 (wild cards) peuvent être jouées à tout moment.

L'objectif immédiat est d'être le premier à ne plus avoir de cartes, les cartes du joueur adverse étant alors totalisées et ajoutées à son score selon les règles suivantes :

cartes numérotées 0-9 = valeur du numéro de face

Inverse, Skip, +2 = 20pts

Wild, Wild +4 = 50pts

Le premier joueur qui atteint 100 perd le jeu

**Gestion des victoires et du score**

Quand je gagne une partie, score de l’ordi augmente

Quand je perds une partie, score de l’ordi diminue

A chaque fin de partie, le perdant récolte le score cumulé des valeurs des cartes de son adversaire

Le premier joueur qui atteint 100 perd la partie.

D’où la stratégie de jouer les jocker tirage au dernier moment pour remporter le plus de points au moment du gain.

Il peut donc arriver que le joueur ou l’ordi gagne le jeu en 1 seule et unique partie.

**Gestion des cartes et du deck de cartes**

Initialisation du jeu de cartes en associant :

* Chiffres à une couleur
* Chiffres au numéro de cartes

Les cartes sont déployées comme des objets avec la création d’une classe « carte »

Une image contenant texte

Description générée automatiquement

Les cartes numérotées sont directes :

rouge8 = {

valeur : 8,

point : 8,

color : 'red',

changeTurn : true,

drawValue : 0

}

Pour des raisons de logique, les cartes d'action se verront attribuer une valeur, par ordre croissant de danger pour l’ordi en cas de défaite :

greenReverse = {

valeur : 10,

point : 20,

color : 'green',

changeTurn : false,

drawValue : 0

}

orangeSkip = {

valeur : 11,

point : 20,

couleur : 'jaune',

changeTurn : false,

drawValue : 0

}

blueDraw2 = {

valeur : 12,

point : 20,

couleur : 'bleu',

changeTurn : true,

drawValue : 2

}

wild = {

valeur : 13,

point : 50,

color : 'any',

changeTurn : true,

drawValue : 0

}

wild4 = {

valeur : 14,

point : 50,

color : 'any',

changeTurn : true,

drawValue : 4

}

Le gameController utilisera les propriétés changeTurn et drawValue pour déterminer de quel tour il s'agit et si des cartes doivent être tirées ou non.

* Ces cartes crées sont ensuite appelées pour construire le deck de pioche
* MAJ/Mélange du deck avec Méthode Fisher Yates

Méthode du Quick Sort

Les 3 dernières lignes échangent les cartes en i-ième et en j-ième position tout comme dans notre premier algorithme naïf, mais ici on ne choisit la carte à échanger que parmi celles qui n’ont pas encore été traitées. Et pour des raisons de simplicité du code, on mélange le tableau “depuis la fin”.

Et le plus beau est que sa [complexité algorithmique](http://www.wikipedia.org/search-redirect.php?language=fr&go=Go&search=complexit%C3%A9+algorithmique) est proportionnelle à N, ce qui est in exemple de plus de l’universalité du principe de Murphy de la thermodynamique de la tartine beurrée qui dit qu’il est toujours plus facile de générer du désordre que de l’ordre.

D’autre part, si vous devez brasser beaucoup de données de manière irréversible, je ne sais pas moi, pour une application de vote par internet par exemple, souvenez-vous de Fisher-Yates. Et utilisez un [vrai générateur de nombres aléatoires](http://www.idquantique.com/component/content/article/9.html) , car un [générateur de nombres pseudo-aléatoires](http://www.wikipedia.org/search-redirect.php?language=fr&go=Go&search=g%C3%A9n%C3%A9rateur+de+nombres+pseudo-al%C3%A9atoires) peut non seulement souffrir de biais, mais surtout être reproductible à partir de la “graine” et permettre ainsi d’inverser le brassage.

algorithme pour générer une permutation aléatoire d'un ensemble fini, c'est-à-dire pour mélanger un ensemble d'objets.

* **MAJ du deck une fois fin de partie ou du jeu afin que les parties ne se ressemblent pas toutes.**

**Evolution du Score**

Maj des points après chaque partie en fonction des points sur les cartes restantes

Une image contenant texte

Description générée automatiquement

**LOGIQUE DE L’ORDI**

L’ordi aura deux tableaux dont elle assurera le suivi :

ordiMain = []

playable = []

En se basant sur la dernière carte jouée et ses propriétés, l’ordi va parcourir en boucle son tableau ordiMain qui constitue la liste de cartes à sa disposition. Toute carte qui correspond à la valeur ou à la couleur de la dernière carte jouée sera placée dans le tableau playableCards avec les jokers que l’ordi peut détenir.

Ce tableau playable construit est mis à jour après chaque tour du joueur en fonction de la carte jouée (même couleur ou même chiffre ou carte d’actions). Il s’agit d’un tableau temporaire.

Viens ensuite 3 choix possibles pour l’ordi :

* Si aucune carte dans « playable » en adéquation avec la dernière carte jouée , il pioche
* Si il a une carte (playable) en adéquation avec la dernière carte jouée, il l’a joue
* Si il a plusieurs cartes, il applique une stratégie :

Puisqu'une partie du plaisir et de la stratégie consiste à savoir quand jouer ses cartes d'action, l'ordi déterminera sa stratégie de manière aléatoire à chaque tour, à l'aide d'une variable Math.Random(). Si la variable aléatoire est supérieure à 0,5, l'ordi donnera la priorité aux cartes d'action afin de maintenir son score de perte à un niveau bas. Si la variable aléatoire est inférieure à 0,5, l'ordi conservera ses cartes d'action pour un tour ultérieur et jouera plutôt des cartes de chiffres. Il y aura également une logique permettant d'ignorer la randomisation lorsque le joueur atteint un certain nombre de cartes, auquel cas l'unité centrale donnera la priorité aux cartes d'action.

**Choix de jeux de l’ordi**

Liste de cartes à sa dispo 🡪 Construit tableau playable après chaque tour du joueur en fonction de la carte jouée (même couleur ou même chiffre ou carte d’actions) 🡪 3 choix ensuite :

* Si aucune carte , il pioche
* Si il a une carte (playable), il l’a joue
* Si il a plusieurs cartes, il applique la stratégie

**EXPÉRIENCE DU JOUEUR**

De la même manière que l'ordinateur gardera la trace des cartes qu'il peut jouer, le contrôleur de jeu fera de même pour le joueur. Si le joueur clique sur une carte non valide, un message s'affiche pour l'en informer.

Ces messages et l'écran "Game Over" seront les seules invites à l'écran afin de minimiser les distractions et de permettre au déroulement du jeu d'occuper le devant de la scène.

(Dans un effort de protection contre les clics involontaires, il pourrait y avoir un message "Êtes-vous sûr ?" si le joueur clique sur la pile de tirage alors qu'il tient des cartes jouables).

L'objectif est de créer une boucle de jeu esthétiquement agréable, minimale mais satisfaisante, qui soit relaxante et divertissante et à laquelle, espérons-le, les utilisateurs voudront jouer encore et encore.

Transitions CSS

<https://coursework.vschool.io/css-transitions/>

**Hover** : lorsque que le curseur de la souris passe au-dessus d’un boutton ou d’un élément, celui-ci se me en subrillance.

+ Image mise en avant par-dessus le curseur grâce à la fonction transition «**all 1s ease-in-out** »

Une image contenant texte

Description générée automatiquement

+ mouvement de cartes

A l’aide de la fonction « **translate** »

Une image contenant texte

Description générée automatiquement

**AXES D’amélioration**

Refactorisation du code 🡪 J'aurais aimé avoir plus de temps pour refactoriser et appairer mon code. J'ai fait du mieux que j'ai pu, mais j'ai construit un peu un labyrinthe. En raison des différentes façons dont la logique de l’ordi et du joueur sont structurées et de la façon dont les mains sont conservées et liées au DOM, il a été un peu difficile de créer des fonctions qui fonctionnent pour les deux joueurs.

Le seul problème qui me préoccupe (jeu de mots) est que de temps en temps, après que l'utilisateur ait joué un joker et choisi une nouvelle couleur, le processeur a tendance à jouer plusieurs cartes en même temps. Ce n'est pas de la triche - lorsque vous vérifiez les journaux de la console, il joue généralement un saut ou une carte inversée, suivie d'une carte numérique, mais tout se passe en même temps et on dirait que l’ordi joue 2 ou 3 cartes en même temps.

* Tendance à ne pas avoir les cartes pour jouer et donc de piocher une carte, une par une pour le joueur et l’ordi allant jusqu’à 5 tours
* Tendance de l’ordi a être agressif lorsqu’il a des +4 ou +2 à sa disposition.

Règles du Uno à respecter entièrement dans le jeu (exemple d’un boutton Uno à cliquer dessus par le joueur

Mode multi-joueur

**Sources**

**Banque images uno**

<https://www.pngfind.com/mpng/iJbmTbR_uno-cards-png-cards-are-in-an-uno/>

**Gestion deck cartes**

<https://w3collective.com/random-playing-card-javascript/>

Sons :

<https://www.soundjay.com/card-sounds-1.html>

<https://www.pond5.com/fr/search?kw=uno&media=sfx>

<https://freesound.org/people/Zomora/downloaded_sounds/$>

Codes inspiration

<https://github.com/YukiLand/Memory-game-Javascript>

<https://github.com/AlexisAmand/The-Gasp-JS>