console.log('uno!')

//DOM et variables globales

const ordiMainDom = document.querySelector('.ordi-main')

const joueurMainDom = document.querySelector('.player-hand')

const ordiScoreDom = document.querySelector('#ordi-score')

const joueurScoreDom = document.querySelector('#joueur-score')

const jeuPileDom = document.querySelector('.joueur-pile')

const piochePileDom = document.querySelector('.draw-pile')

const joueurUno = document.querySelector('.player-animation')

const ordiUno = document.querySelector('.cpu-animation')

// hand arrays

const ordiMain = []

const joueurMain = []

const deck = []

let jouerPile

let ordiScore = 0

let joueurScore = 0

// Définition des variables du gameplay

let tourJoueur = true

let gameOn = true

let choixCouleurEstOuvert = false

// Minuteur en milisecondes en fonction de la taille de main de l'ordi

// Math.floor(x) renvoie le plus grand entier qui est inférieur ou égal à un nombre x

let OrdiDelai = Math.floor((Math.random() \* ordiMain.length \* 200) + 1500)

// Score limite --> Si il est atteint, la personne a perdu le jeu complet

let gameOver = 100

//#endregion

//pour chargement plus rapide

const imgPreLoad = []

let preLoaded = false

// Initialisation du jeu de cartes

const preLoadImgs = () => {

// Pour chaque couleur (au nombre de 4)

for (let i = 0; i <= 3; i++) {

let color

// Association couleur et numéro

if (i === 0) color = 'red'

if (i === 1) color = 'green'

if (i === 2) color = 'blue'

if (i === 3) color = 'yellow'

// Pour chacune des cartes

for (let n = 0; n <= 14; n++) {

let img = new Image()

// Lien relatif menant aux images successives

img.src = 'images/' + color + i + '.png'

imgPreLoad.push(img)

}

}

for (let i = 0; i < imgPreLoad.length; i++) {

// On ajoute les images dans dans notre jeu

jeuPileDom.appendChild(imgPreLoad[i])

jeuPileDom.innerHTML = ''

}

}

// Initialisation des variables audios

const melange = new Audio('audio/shuffle.wav')

const jouerCarte = new Audio('audio/playCardNew.wav')

const jouerCarte2 = new Audio('audio/playCard2.wav')

const piocheCarte = new Audio('audio/drawCard.wav')

const gagnepartie = new Audio('audio/winRound.wav')

const gagnejeu = new Audio('audio/winGame.wav')

const partieperdue = new Audio('audio/lose.wav')

const plusCardFX = new Audio('audio/plusCard.wav')

const uno = new Audio('audio/uno.wav')

const boutoncouleur = new Audio('audio/colorButton.wav')

const JouerEncore = new Audio('audio/playAgain.wav')

const SonJouerCarte = () => {

jouerCarte2.play()

}

// Carte et du DECK MANAGEMENT

// On utilisera les propriétés changeTour et drawValue pour déterminer de quel tour il s'agit et si des cartes doivent être tirées ou non.

class Carte {

constructor(rgb, valeur, points, changeTour, drawValue, imgSrc) {

this.color = rgb

this.valeur = valeur

this.points = points

this.changeTour = changeTour

this.drawValue = drawValue

this.src = imgSrc

this.joueParjoueur = false

}

}

// Création des cartes avec appel des objets cartes

const creerCarte = (rgb, color) => {

for (let i = 0; i <= 14; i++) {

// nombre de cartes

if (i === 0) {

deck.push(new Carte(rgb, i, i, true, 0, 'images/' + color + i + '.png'))

}

else if (i > 0 && i <= 9) {

deck.push(new Carte(rgb, i, i, true, 0, 'images/' + color + i + '.png'))

deck.push(new Carte(rgb, i, i, true, 0, 'images/' + color + i + '.png'))

}

// reverse/skip cartes (passe son tour/sens inverse

else if (i === 10 || i === 11) {

deck.push(new Carte(rgb, i, 20, false, 0, 'images/' + color + i + '.png'))

deck.push(new Carte(rgb, i, 20, false, 0, 'images/' + color + i + '.png'))

}

// +2 cartes

else if (i === 12) {

deck.push(new Carte(rgb, i, 20, false, 2, 'images/' + color + i + '.png'))

deck.push(new Carte(rgb, i, 20, false, 2, 'images/' + color + i + '.png'))

}

else if (i === 13) {

deck.push(new Carte('any', i, 50, true, 0, 'images/wild' + i + '.png'))

}

else {

deck.push(new Carte('any', i, 50, false, 4, 'images/wild' + i + '.png'))

}

}

}

// Création du deck à partir des cartes crées précédemment

const creerDeck = () => {

// détruit le deck précédent

deck.length = 0

// creéer un nouveau deck

for (let i = 0; i <= 3; i++){

if (i === 0) {

creerCarte('rgb(255, 6, 0)', 'red')

}

else if (i === 1) {

creerCarte('rgb(0, 170, 69)', 'green')

}

else if (i === 2) {

creerCarte('rgb(0, 150, 224)', 'blue')

}

else {

creerCarte('rgb(255, 222, 0)', 'yellow')

}

}

}

// Mélange du deck après l'avoir créer

const MelangeDeck = (deck) => {

// Fisher-Yates Method - https://www.frankmitchell.org/2015/01/fisher-yates/

// Mélange les cartes du jeu

for (let i = deck.length - 1; i > 0; i--) {

deck[i].joueParjoueur = false

let j = Math.floor(Math.random() \* (i + 1))

let temp = deck[i]

deck[i] = deck[j]

deck[j] = temp

}

melange.play()

}

// Comportement gameplay

const DistribCartes = () => {

for (let i = 0; i < 7; i++) {

// distribue les cartes mélangées dans les tableaux ordi et joueurs respectifs

ordiMain.push(deck.shift())

joueurMain.push(deck.shift())

// ajoute les cartes dans le DOM

const ordiCarte = document.createElement('img')

ordiCarte.setAttribute('src', 'images/back.png')

ordiCarte.setAttribute('class', 'cpu')

ordiMainDom.appendChild(ordiCarte)

const CarteJeu = document.createElement('img')

CarteJeu.setAttribute('src', joueurMain[i].src)

CarteJeu.setAttribute('class', 'player')

// Attribuer aux Cartes un id = leur index dans le tableau joueurMain.

//afin de référencer le bon objet Carte

CarteJeu.setAttribute('id', i)

joueurMainDom.appendChild(CarteJeu)

}

}

const DebutePileJeu = () => {

const jouerCarte = document.createElement('img')

// trouve la 1ère carte qui n'est pas unecarte d'action

for (let i = 0; i < deck.length; i++) {

if (deck[i].color !== "any" && deck[i].valeur <= 9) {

// commence jouerPile tableau avec la première carte valide

jouerPile = deck.splice(i, 1)

break

}

}

// associe jouerCarte à l'image correcte

jouerCarte.setAttribute('src', jouerPile[0].src)

// joue la carte de la jouerPile

jeuPileDom.appendChild(jouerCarte)

}

// Mise à jour du deck après une fin de partie ou de jeu

const nouvelleMain = () => {

gameOn = true

// maj la main de l'ordi, du joueur et la play pile

ordiMainDom.innerHTML = ''

ordiMain.length = 0

joueurMainDom.innerHTML = ''

joueurMain.length = 0

jeuPileDom.innerHTML = ''

// créer un nouveau deck

creerDeck()

// mélange le deck

MelangeDeck(deck)

//distribution des cartes et mise de la première

DistribCartes()

// Pose la première carte à jouer qui n'est pas une carte d'action.

DebutePileJeu()

if (choixCouleurEstOuvert) CacheChoixCouleur()

}

// Mise à jour de la pile de jeu

const updatejeuPileDom = () => {

jeuPileDom.innerHTML = ''

// ajoute une carte jouée dans jouerPile

const newCardImg = document.createElement('img')

const imgSrc = jouerPile[jouerPile.length - 1].src

newCardImg.setAttribute('src', imgSrc)

jeuPileDom.appendChild(newCardImg)

}

// Mise à jour de la main

const majMain = (handToUpdate) => {

let domToUpdate, cardClass;

if (handToUpdate === ordiMain) {

domToUpdate = ordiMainDom

cardClass = 'cpu'

if (ordiVisible) ordiVisible = false

}

else {

domToUpdate = joueurMainDom

cardClass = 'player'

}

// efface le domaine sélectionné

domToUpdate.innerHTML = ''

// maj dom

for (let i = 0; i < handToUpdate.length; i++) {

let src

if (domToUpdate === ordiMainDom) {

src = 'images/back.png'

}

else {

src = handToUpdate[i].src

}

const majCartes = document.createElement('img')

majCartes.setAttribute('src', src)

majCartes.setAttribute('class', cardClass)

// mise à jour des ID pour correspondre aux index de joueurMain

majCartes.setAttribute('id', i)

domToUpdate.appendChild(majCartes)

}

// Fin jeu/partie : empêche l'élément dom de ne plus exister lorsque la main est vide

if (handToUpdate.length === 0) {

const majCartes = document.createElement('img')

majCartes.setAttribute('src', 'images/empty.png')

majCartes.setAttribute('class', 'empty')

// mise à jour des ID pour correspondre aux index de joueurMain

domToUpdate.appendChild(majCartes)

}

}

// Tirage de cartes

const tireCarte = (MainObtientCarte) => {

animatetireCarte(MainObtientCarte)

// vérifier si le deck a une carte à tirer

if (deck.length > 0) {

// tirer la carte du haut

const newCard = deck.shift()

MainObtientCarte.push(newCard)

}

else {

// Sinon, nouvelle partie donc nouveau deck --> mélanger jouerPile

MelangeDeck(jouerPile)

for (let i = 0; i <= jouerPile.length - 1; i++) {

// la pile de joueurs mélangée devient le nouveau jeu de cartes.

deck.push(jouerPile[i])

}

// laisse la dernière carte jouée sur le tas de jeu.

jouerPile.length = 1

// tire la plus haute carte du jeu

const newCard = deck.shift()

MainObtientCarte.push(newCard)

}

piocheCarte.play()

setTimeout(() => {

majMain(MainObtientCarte)

}, 500)

}

// Animation tirage carte

const animatetireCarte = (player) => {

let playerClass

if (player === ordiMain) playerClass = 'cpu-draw'

else playerClass = 'player-draw'

const drawCardEl = document.querySelector('#draw-card')

drawCardEl.classList.remove('hidden')

setTimeout(() => {

drawCardEl.classList.add(playerClass)

setTimeout(() => {

drawCardEl.classList.add('hidden')

drawCardEl.classList.remove(playerClass)

clearInterval()

}, 500)

}, 30)

}

const montreUno = (unoMain) => {

unoMain.classList.remove('hidden')

uno.play()

// add shout class

setTimeout(() => {

unoMain.classList.add('shout')

//setTimeout = après x secondes, retirer le son uno

setTimeout(() => {

unoMain.classList.remove('shout')

setTimeout(() => {

unoMain.classList.add('hidden')

console.log('added HIDDEN to', unoMain)

}, 1000)

}, 1000)

}, 10)

}

const afficheChoixCouleur = () => {

// montre le choix de couleur

const colorPicker = document.querySelector('.color-picker')

colorPicker.style.opacity = 1

choixCouleurEstOuvert = true

//Assigner les gestionnaires d'événements aux boutons

document.querySelector('.red').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

choisiCouleur('rgb(255, 6, 0)')

})

document.querySelector('.green').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

choisiCouleur('rgb(0, 170, 69)')

})

document.querySelector('.blue').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

choisiCouleur('rgb(0, 150, 224)')

})

document.querySelector('.yellow').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

choisiCouleur('rgb(255, 222, 0)')

})

}

const choisiCouleur = (rgb) => {

//attribuer la couleur au joker qui se trouve sur le dessus de la pile de jeu

boutoncouleur.play()

jouerPile[jouerPile.length - 1].color = rgb

// cache le choix de couleur

CacheChoixCouleur()

tourJoueur = false;

setTimeout(jouerOrdi, OrdiDelai)}

function CacheChoixCouleur() {

const colorPicker = document.querySelector('.color-picker')

colorPicker.style.opacity = 0

choixCouleurEstOuvert = false

}

const passeOuinversion = () => {

// vérifie si c'est changement de tour ou passe son tour

if (jouerPile[jouerPile.length - 1].changeTour) {

tourJoueur = false

// au tour de l'ordi

setTimeout(jouerOrdi, OrdiDelai)

}

}

// met à jour les noms des joueurs avec le nom de celui dont c'est le tour

//mise en subrillance

const AfficheAuTourDom = () => {

if (tourJoueur) {

document.querySelector('.joueur-score-titre').style.color = 'rgb(100, 150, 150)'

document.querySelector('.ordi-score-titre').style.color = 'rgb(6, 37, 62)'

}

else {

document.querySelector('.joueur-score-titre').style.color = 'rgb(6, 37, 62)'

document.querySelector('.ordi-score-titre').style.color = 'rgb(100, 150, 150)'

}

}

//Fin de la partie et du jeu

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

const CompterPoints = (PerdantMain) => {

// Init à 0

let points = 0

for (const card of PerdantMain) {

points += card.points

}

if (PerdantMain == ordiMain) {

ordiScore += points

}

else {

joueurScore += points

}

}

const MajScores = () => {

// met à jour le score de l'ordinateur

ordiScoreDom.innerHTML = ordiScore

// Si la gagne est pour l'ordi

if (ordiScore < gameOver / 2) ordiScoreDom.style.color = 'rgb(0, 140, 0)' // couleur initiale du score en vert

else ordiScoreDom.style.color = 'rgb(121, 2, 2)' // évolution de la couleur en rouge

// met à jour le score du joueur

joueurScoreDom.innerHTML = joueurScore

// Si la gagne est pour le joueur

if (joueurScore < gameOver / 2) joueurScoreDom.style.color = 'rgb(0, 140, 0)'

else joueurScoreDom.style.color = 'rgb(121, 2, 2)'

}

const VerifGagnant = () => {

// vérifie si personne n'a perdu

if (joueurScore < gameOver && ordiScore < gameOver) {

// si joueur a gagné

if (joueurMain.length === 0) {

gagnepartie.play()

finPartie(joueurMain)

}

// Si ordi a gagné

if (ordiMain.length === 0) {

partieperdue.play()

finPartie(ordiMain)

}

}

// sinon fin du jeu car la personne a atteint le score limite !

else {

finJeu()

}

}

const AfficheOrdiCartes = () => {

if (ordiMain.length >= 1) {

ordiMainDom.innerHTML = ''

for (let i = 0; i < ordiMain.length; i++) {

// retourne les cartes

const ordiCarte = document.createElement('img')

ordiCarte.setAttribute('src', ordiMain[i].src)

ordiCarte.setAttribute('class', 'cpu')

ordiMainDom.appendChild(ordiCarte)

}

}

}

const CacheOrdiCards = () => {

if (ordiMain.length >= 1) {

ordiMainDom.innerHTML = ''

for (let i = 0; i < ordiMain.length; i++) {

// retourne les cartes, vue de dos

const ordiCarte = document.createElement('img')

ordiCarte.setAttribute('src', 'images/back.png')

ordiCarte.setAttribute('class', 'cpu')

ordiMainDom.appendChild(ordiCarte)

}

}

}

const finPartie = (winner) => {

gameOn = false;

tourJoueur = !tourJoueur

if (ordiMain.length > 0) AfficheOrdiCartes()

const finPartieDom = document.querySelector('.end-of-round')

const partieDom = document.querySelector('.round')

// affiche l'élément de fin de tour en fonction du gagnant

finPartieDom.classList.remove('hidden')

if (winner === joueurMain) partieDom.textContent = 'Tu as gagné la partie!'

else partieDom.textContent = 'Lordi a gagné la partie...'

// cache l'élements de fin de partie après 2 sec

setTimeout(() => {

finPartieDom.classList.add('hidden')

tourJoueur = !tourJoueur

nouvelleMain()

if (!tourJoueur) setTimeout(jouerOrdi, OrdiDelai)

}, 3000) // 3 sec d'attente avant la nouvelle partie

}

const finJeu = () => {

gameOn = false;

if (ordiMain.length > 0) AfficheOrdiCartes()

const finDujeuDom = document.querySelector('.end-of-game')

const gameDom = document.querySelector('.jeu')

// affiche l'élément de fin de partie et le format en fonction du gagnant

finDujeuDom.classList.remove('hidden')

// Affichage des bulles

if (joueurScore > gameOver) {

partieperdue.play()

gameDom.textContent = 'Ordinateur a gagné le jeu ... Encore une game ?'

}

else {

gagnejeu.play()

gameDom.textContent = 'Tu as gagné le jeu ! Encore une game ?'

}

// ajoute un listener pour le bouton 'play again'

document.querySelector('.play-again').addEventListener('click', () => {

JouerEncore.play()

// cache l'élément de fin de jeu au clic

finDujeuDom.classList.add('hidden')

joueurScore = 0

ordiScore = 0

MajScores()

tourJoueur = !tourJoueur

nouvelleMain()

if (!tourJoueur) setTimeout(jouerOrdi, OrdiDelai)

})

}

//Logique ordinateur comme une ia

const laisseOrdiTireCarte = () => {

if (jouerPile[jouerPile.length - 1].drawValue > 0) {

// ajoute le nombre de cartes basé sur la valeur de tirage de la dernière carte jouée.

for (let i = 0; i < jouerPile[jouerPile.length - 1].drawValue; i++) {

tireCarte(ordiMain)

}

}

}

// Choix de jeux de l'ordi

const jouerOrdi = () => {

if (!tourJoueur && gameOn) {

// En se basant sur la dernière carte jouée et ses propriétés, lordi va parcourir en boucle

//son tableau ordiMain, et 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 lordi

// peut détenir.

// créer un tableau temporaire playable de cartes jouables basé sur la dernière carte jouée

const playable = determinePlayableCards()

// Si pas de cartes à jouer

if (playable.length === 0) {

// draw card

tireCarte(ordiMain)

// end turn

setTimeout(() => {

// L'ordi a finit son tour, c'est maintenant au tour du joueur

tourJoueur = true

return

}, 500)

}

//Si une carte jouable (présente dans le tableau "playable", il joue la carte)

else if (playable.length === 1) {

setTimeout(jouerOrdiCarte, 300, playable[0])

}

// Si il y a plus d'1 carte jouable

else if (playable.length > 1) {

// application de la stratégie définie précédemment

let cartechoisie = lancerStrategie(playable)

setTimeout(jouerOrdiCarte, 300, cartechoisie)

}

}

//Fonctions spécifiques à l'ordinateur

// Définition tableau "playable" cartes

function determinePlayableCards() {

const playableCards = []

console.log(jouerPile[jouerPile.length - 1])

for (let i = 0; i < ordiMain.length; i++) {

// si même couleur ou même chiffre

if (ordiMain[i].color === jouerPile[jouerPile.length - 1].color || ordiMain[i].valeur === jouerPile[jouerPile.length - 1].valeur || ordiMain[i].color === 'any' || jouerPile[jouerPile.length - 1].color === 'any') {

let validCard = ordiMain.splice(i, 1)

// Le tableau se remplit avec les cartes qui correspondent à ce critère

playableCards.push(validCard[0])

}

}

return playableCards

}

function lancerStrategie(playable) {

let carteIndex

// Détermination de la stratégie de manière aléatoire à chaque tour

// Si la variable aléatoire est supérieure à 0,5, l'ordi donnera la priorité aux cartes

// d'action (jockers) 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.

// Initialisation aléatoire de la stratégie

let strategie = Math.random()

// Si la variable aléatoire est supérieure à 0,5

if (jouerPile.length > 2 && (strategie > 0.5 || joueurMain.length < 3 || ordiMain.length > (joueurMain.length \* 2) || (jouerPile[jouerPile.length - 1].joueParjoueur === true && jouerPile[jouerPile.length - 1].drawValue > 0) || (jouerPile[jouerPile.length - 2].joueParjoueur === true && jouerPile[jouerPile.length - 1].drawValue > 0))) {

// Lordi choisis et priorise la carte chiffre avec le nombre de points le plus élevé

let PlusHauteValeur = 0

for (let i = 0; i < playable.length; i++){

if (playable[i].valeur > PlusHauteValeur) {

PlusHauteValeur = playable[i].valeur

carteIndex = i

}

}

// joue la carte déterminée par stratégie

// retire la carte depuis playable

cartechoisie = playable.splice(carteIndex, 1)

// retourne playable de ordiMain

returnPlayablesToHand()

}

// Si la variable aléatoire est inférieure à 0,5

else {

// sinon priorise les cartes jokers comme la couleur ou le nombre de cartes

let plusFaibleValeur = 14

for (let i = 0; i < playable.length; i++){

if (playable[i].valeur < plusFaibleValeur) {

plusFaibleValeur = playable[i].valeur

carteIndex = i

}

}

// joue la carte déterminée par stratégie

// retire la carte depuis playable

cartechoisie = playable.splice(carteIndex, 1)

returnPlayablesToHand()

}

return cartechoisie[0]

function returnPlayablesToHand() {

if (playable.length > 0) {

for (const card of playable) {

ordiMain.push(card)

}

}

}

}

function jouerOrdiCarte(cartechoisie) {

//carte aléatoire de ordiMainDom

const ordiDomCards = ordiMainDom.childNodes

ordiDomCards[Math.floor(Math.random() \* ordiDomCards.length)].classList.add('cpu-play-card')

SonJouerCarte()

// debugger

setTimeout(() => {

jouerPile.push(cartechoisie)

// maj jeuPileDom

updatejeuPileDom()

// vérifie si lordi a joué au hasard

if (jouerPile[jouerPile.length - 1].color === 'any' && jouerPile[jouerPile.length - 1].drawValue === 0 && jouerPile[jouerPile.length - 1].joueParjoueur === false) {

ChoisirCouleurApresWild()

}

// vérifie la taille de la main de l'ordi et maj ordiMainDom

if (ordiMain.length >= 1) {

majMain(ordiMain)

if (ordiMain.length === 1) {

montreUno(ordiUno)

}

}

// si fin du tour

else {

majMain(ordiMain)

setTimeout(() => {

CompterPoints(joueurMain)

MajScores()

VerifGagnant()

}, 1200)

}

// si lordi tire une carte

if (cartechoisie.drawValue > 0) {

console.log('Lordi a joué +' + cartechoisie.drawValue + ' carte !')

PenalitéPlusQuatreCarte()

setTimeout(() => {

for (let i = 0; i < cartechoisie.drawValue; i++) {

tireCarte(joueurMain)

}

VerifieChangementTour()

},1000)

}

else setTimeout(VerifieChangementTour, 500)

}, 500)

function VerifieChangementTour() {

if (cartechoisie.changeTour) {

// si changement de tour, tourJoueur = true

tourJoueur = true

return

}

else {

// pas de changment, au tour de lordi

setTimeout(jouerOrdi, OrdiDelai)

}

}

}

function ChoisirCouleurApresWild() {

// 4 couleurs possibles

const couleurs = ['rgb(255, 6, 0)', 'rgb(0, 170, 69)', 'rgb(0, 150, 224)', 'rgb(255, 222, 0)']

const CouleursDansMain = [0, 0, 0, 0]

// Le processeur vérifie combien il a de cartes de chaque couleur.

for (const card of ordiMain) {

// itérations pour chaque couleur en comparant les deux listes

if (card.color === couleurs[0]) CouleursDansMain[0]++

if (card.color === couleurs[1]) CouleursDansMain[1]++

if (card.color === couleurs[2]) CouleursDansMain[2]++

if (card.color === couleurs[3]) CouleursDansMain[3]++

}

// trouver l'indice de la valeur maximale

let indexOfMax = CouleursDansMain.indexOf(Math.max(...CouleursDansMain))

// carte wild et choix des couleurs

const wildCarteDom = jeuPileDom.childNodes[0]

wildCarteDom.style.border = '5px solid ' + couleurs[indexOfMax]

wildCarteDom.style.width = '105px'

jouerPile[jouerPile.length - 1].color = couleurs[indexOfMax]

}

}

// Wild +4

const PenalitéPlusQuatreCarte = () => {

plusCardFX.play()

jeuPileDom.classList.add('shout')

setTimeout(() => {

jeuPileDom.classList.remove('shout')

}, 1000)

}

const JouerJoueurCarte = (index) => {

let CarteAJouer = joueurMain.splice(index, 1)

CarteAJouer[0].joueParjoueur = true

jouerPile.push(CarteAJouer[0])

// vide la pile de jeux

updatejeuPileDom()

}

///////Principales fonctions du jeu - MAIN ////////

// Jeu

const startGame = () => {

if (!preLoaded) {

preLoadImgs()

preLoaded = true

}

joueurScore = 0

ordiScore = 0

listenerKeyMode()

setInterval(AfficheAuTourDom, 100)

nouvelleMain()

MajScores()

if (!tourJoueur) setTimeout(jouerOrdi, OrdiDelai)

// Listeners sur joueurMainDom et piochePileDom au clic

// joueurMainDom

joueurMainDom.addEventListener('click', (event) => {

if (tourJoueur && !choixCouleurEstOuvert && event.target.getAttribute('id')) {

const DerniereCarteDom = jeuPileDom.childNodes[0]

if (DerniereCarteDom.style !== '100px') {

DerniereCarteDom.style.width = '100px'

DerniereCarteDom.style.border = 'none'

}

// utiliser l'ID de la cible pour trouver l'objet carte dans le tableau

let index = parseInt(event.target.getAttribute('id'))

// si la valeur ou la couleur correspondent à la carte de la pile

if (joueurMain[index].valeur === jouerPile[jouerPile.length - 1].valeur || joueurMain[index].color === jouerPile[jouerPile.length - 1].color || joueurMain[index].color === 'any' || jouerPile[jouerPile.length - 1].color === 'any') {

// carte animée au clic

SonJouerCarte()

event.target.classList.add('play-card')

setTimeout(() => {

JouerJoueurCarte(index)

// appelle cpuTurn pour ajouter des cartes s'il y en a à ajouter

laisseOrdiTireCarte()

// vérifié la taille de joueurMain et maj DOM

if (joueurMain.length >= 1) {

majMain(joueurMain)

// Si plus qu'une carte dans sa main --> Uno

if (joueurMain.length === 1) montreUno(joueurUno)

}

// FIn de la partie --> maj points et scores et on définit le gagnant

else {

majMain(joueurMain)

setTimeout(() => {

CompterPoints(ordiMain)

MajScores()

VerifGagnant()

}, 1200)

}

//vérifie si carte tirée et carte "wild" (choix des couleurs)

if (jouerPile[jouerPile.length - 1].color === 'any' && jouerPile[jouerPile.length - 1].drawValue === 0 && jouerPile[jouerPile.length - 1].joueParjoueur) {

// nouvelle couleur

afficheChoixCouleur()

return

}

passeOuinversion();

}, 1000)

}

}

})

piochePileDom.addEventListener('click', () => {

if (tourJoueur && !choixCouleurEstOuvert) {

tireCarte(joueurMain)

setTimeout(() => {

tourJoueur = false;

setTimeout(jouerOrdi, OrdiDelai)

}, 500)

}

})

}

//#endregionlet ordiVisible = false

const listenerKeyMode = () => {

document.addEventListener('keydown', event => {

const key = event.key.toLowerCase()

console.log(key)

})

}

startGame()

//// JEU DU UNO - Développement Java Script

//DOM et variables globales

const ordiMainDom = document.querySelector('.ordi-main')

const joueurMainDom = document.querySelector('.player-hand')

const ordiScoreDom = document.querySelector('#ordi-score')

const joueurScoreDom = document.querySelector('#joueur-score')

const jouerPileDom = document.querySelector('.joueur-pile')

const piochePileDom = document.querySelector('.draw-pile')

const joueurUno = document.querySelector('.player-animation')

const ordiUno = document.querySelector('.cpu-animation')

// Tableaux des mains de joueur et de l'ordinateur

const ordiMain = []

const joueurMain = []

const deck = []

let jouerPile

let ordiScore = 0

let joueurScore = 0

// Définition des variables du gameplay

let tourJoueur = true

let gameOn = true

let choixCouleurEstOuvert = false

// Minuteur en milisecondes en fonction de la taille de main de l'ordi

// Math.floor(x) renvoie le plus grand entier qui est inférieur ou égal à un nombre x

let ordiDelai = Math.floor((Math.random() \* ordiMain.length \* 200) + 1500)

// Score limite --> Si il est atteint, la personne a perdu le jeu complet

let gameOver = 100

//pour chargement plus rapide

const imgPreLoad = []

let preLoaded = false

// Initialisation du jeu de cartes

const preLoadImgs = () => {

// Pour chaque couleur (au nombre de 4)

for (let i = 0; i <= 3; i++) {

let color

// Association couleur et numéro

if (i === 0) color = 'red'

if (i === 1) color = 'green'

if (i === 2) color = 'blue'

if (i === 3) color = 'yellow'

// Pour chacune des cartes

for (let n = 0; n <= 14; n++) {

let img = new Image()

// Lien relatif menant aux images successives

img.src = 'images/' + color + i + '.png'

imgPreLoad.push(img)

}

}

for (let i = 0; i < imgPreLoad.length; i++) {

// On ajoute les images dans dans notre jeu

jouerPileDom.appendChild(imgPreLoad[i])

jouerPileDom.innerHTML = ''

}

}

// Initialisation des variables audios

const melange = new Audio('audio/shuffle.wav')

const jouerCarte = new Audio('audio/playCardNew.wav')

const jouerCarte2 = new Audio('audio/playCard2.wav')

const piocheCarte = new Audio('audio/drawCard.wav')

const gagnePartie = new Audio('audio/winRound.wav')

const gagneJeu = new Audio('audio/winGame.wav')

const partiePerdue = new Audio('audio/lose.wav')

const plusCarteFX = new Audio('audio/plusCard.wav')

const uno = new Audio('audio/uno.wav')

const couleurBoutton = new Audio('audio/colorButton.wav')

const jouerEncore = new Audio('audio/playAgain.wav')

const SonJouerCarte = () => {

jouerCarte2.play()

}

// Carte et du DECK MANAGEMENT

// On utilisera les propriétés changeTour et drawValue pour déterminer de quel tour il s'agit et si des cartes doivent être tirées ou non.

class Card {

constructor(rgb, valeur, points, changeTour, drawValue, imgSrc) {

this.color = rgb

this.valeur = valeur

this.points = points

this.changeTour = changeTour

this.drawValue = drawValue

this.src = imgSrc

this.joueParjoueur = false

}

}

// Création des cartes avec appel des objets cartes

const creerCarte = (rgb, color) => {

for (let i = 0; i <= 14; i++) {

// nombre de cartes

if (i === 0) {

deck.push(new Card(rgb, i, i, true, 0, 'images/' + color + i + '.png'))

}

else if (i > 0 && i <= 9) {

deck.push(new Card(rgb, i, i, true, 0, 'images/' + color + i + '.png'))

deck.push(new Card(rgb, i, i, true, 0, 'images/' + color + i + '.png'))

}

// reverse/skip cartes (passe son tour/sens inverse)

else if (i === 10 || i === 11) {

deck.push(new Card(rgb, i, 20, false, 0, 'images/' + color + i + '.png'))

deck.push(new Card(rgb, i, 20, false, 0, 'images/' + color + i + '.png'))

}

// +2 cartes

else if (i === 12) {

deck.push(new Card(rgb, i, 20, false, 2, 'images/' + color + i + '.png'))

deck.push(new Card(rgb, i, 20, false, 2, 'images/' + color + i + '.png'))

}

else if (i === 13) {

deck.push(new Card('any', i, 50, true, 0, 'images/wild' + i + '.png'))

}

else {

deck.push(new Card('any', i, 50, false, 4, 'images/wild' + i + '.png'))

}

}

}

// Création du deck à partir des cartes crées précédemment

const creerDeck = () => {

// détruit le deck précédent

deck.length = 0

// creéer un nouveau deck

// EN fonction du numéro associé à la couleur

for (let i = 0; i <= 3; i++){

if (i === 0) {

creerCarte('rgb(255, 6, 0)', 'red')

}

else if (i === 1) {

creerCarte('rgb(0, 170, 69)', 'green')

}

else if (i === 2) {

creerCarte('rgb(0, 150, 224)', 'blue')

}

else {

creerCarte('rgb(255, 222, 0)', 'yellow')

}

}

}

// Mélange du deck après l'avoir crée

const MelangerDeck = (deck) => {

// Fisher-Yates Method - https://www.frankmitchell.org/2015/01/fisher-yates/

// Mélange les cartes du jeu

for (let i = deck.length - 1; i > 0; i--) {

deck[i].joueParjoueur = false

let j = Math.floor(Math.random() \* (i + 1))

let temp = deck[i]

deck[i] = deck[j]

deck[j] = temp

}

console.log(deck, 'shuffled') // TODO: remove

melange.play()

}

// Comportement gameplay

const distribCarte = () => {

for (let i = 0; i < 7; i++) {

// distribue les cartes mélangées dans les tableaux ordi et joueurs respectifs

ordiMain.push(deck.shift())

joueurMain.push(deck.shift())

// ajoute les cartes dans le DOM

const ordiCarte = document.createElement('img')

ordiCarte.setAttribute('src', 'images/back.png')

ordiCarte.setAttribute('class', 'cpu')

ordiMainDom.appendChild(ordiCarte)

const carteJeu = document.createElement('img')

carteJeu.setAttribute('src', joueurMain[i].src)

carteJeu.setAttribute('class', 'player')

// Attribuer aux Cartes un id = leur index dans le tableau joueurMain.

//afin de référencer le bon objet Carte

carteJeu.setAttribute('id', i)

joueurMainDom.appendChild(carteJeu)

}

}

const DebuteJeuPile = () => {

const jouerCarte = document.createElement('img')

// trouve la 1ère carte qui n'est pas unecarte d'action

for (let i = 0; i < deck.length; i++) {

if (deck[i].color !== "any" && deck[i].valeur <= 9) {

// commence jouerPile tableau avec la première carte valide

jouerPile = deck.splice(i, 1)

break

}

}

// associe jouerCarte à l'image correcte

jouerCarte.setAttribute('src', jouerPile[0].src)

// joue la carte de la jouerPile

jouerPileDom.appendChild(jouerCarte)

}

// Mise à jour du deck après une fin de partie ou de jeu

const nouvelleMain = () => {

gameOn = true

// maj la main de l'ordi, du joueur et la play pile

ordiMainDom.innerHTML = ''

ordiMain.length = 0

joueurMainDom.innerHTML = ''

joueurMain.length = 0

jouerPileDom.innerHTML = ''

// créer un nouveau deck

creerDeck()

// mélange le deck

MelangerDeck(deck)

//distribution des cartes et mise de la première

distribCarte()

// Pose la première carte à jouer qui n'est pas une carte d'action.

DebuteJeuPile()

if (choixCouleurEstOuvert) cacheChoixCouleur()

}

// Mise à jour de la pile de jeu

const majJouerPileDom = () => {

jouerPileDom.innerHTML = ''

// ajoute une carte jouée dans jouerPile

const newCarteImg = document.createElement('img')

const imgSrc = jouerPile[jouerPile.length - 1].src

newCarteImg.setAttribute('src', imgSrc)

jouerPileDom.appendChild(newCarteImg)

}

// Mise à jour de la main

const majMain = (mainAMaj) => {

let domAMaj, carteClass;

if (mainAMaj === ordiMain) {

domAMaj = ordiMainDom

carteClass = 'cpu'

if (ordiVisible) ordiVisible = false

}

else {

domAMaj = joueurMainDom

carteClass = 'player'

}

// efface le domaine sélectionné

domAMaj.innerHTML = ''

// maj dom

for (let i = 0; i < mainAMaj.length; i++) {

let src

if (domAMaj === ordiMainDom) {

src = 'images/back.png'

}

else {

src = mainAMaj[i].src

}

const majCarte = document.createElement('img')

majCarte.setAttribute('src', src)

majCarte.setAttribute('class', carteClass)

// mise à jour des ID pour correspondre aux index de joueurMain

majCarte.setAttribute('id', i)

domAMaj.appendChild(majCarte)

}

// Fin jeu/partie : empêche l'élément dom de ne plus exister lorsque la main est vide

if (mainAMaj.length === 0) {

const majCarte = document.createElement('img')

majCarte.setAttribute('src', 'images/empty.png')

majCarte.setAttribute('class', 'empty')

// mise à jour des ID pour correspondre aux index de joueurMain

domAMaj.appendChild(majCarte)

}

}

// Tirage de cartes

const tireCarte = (mainObtientCarte) => {

animateTireCarte(mainObtientCarte)

// vérifier si le deck a une carte à tirer

if (deck.length > 0) {

// tirer la carte du haut

const newCard = deck.shift()

mainObtientCarte.push(newCard)

}

else {

// Sinon, nouvelle partie donc nouveau deck --> mélanger jouerPile

MelangerDeck(jouerPile)

for (let i = 0; i <= jouerPile.length - 1; i++) {

// la pile de joueurs mélangée devient le nouveau jeu de cartes.

deck.push(jouerPile[i])

}

// laisse la dernière carte jouée sur le tas de jeu.

jouerPile.length = 1

// tire la plus haute carte du jeu

const newCard = deck.shift()

mainObtientCarte.push(newCard)

}

distribCarte.play()

setTimeout(() => {

majMain(mainObtientCarte)

}, 500)

}

// Animation tirage carte

const animateTireCarte = (player) => {

let joueurClass

//Si c'est l'ordi, on récupére les infos propres aux cpu

if (player === ordiMain) joueurClass = 'cpu-draw'

//Sinon c'est le joueur

else joueurClass = 'player-draw'

const tireCarteEl = document.querySelector('#draw-card')

tireCarteEl.classList.remove('hidden')

setTimeout(() => {

tireCarteEl.classList.add(joueurClass)

setTimeout(() => {

tireCarteEl.classList.add('hidden')

tireCarteEl.classList.remove(joueurClass)

clearInterval()

}, 500)

}, 30)

}

const afficheUno = (unoMain) => {

// supprime la classe "hidden" de la div player-uno

unoMain.classList.remove('hidden')

uno.play()

setTimeout(() => {

unoMain.classList.add('shout')

//setTimeout = après x secondes, retirer le son uno

setTimeout(() => {

unoMain.classList.remove('shout')

setTimeout(() => {

unoMain.classList.add('hidden')

}, 1000)

}, 1000)

}, 10)

}

// Gestion du choix des couleurs

const afficheChoixCouleur = () => {

// montre le choix de couleur

const choixCouleur = document.querySelector('.color-picker')

choixCouleur.style.opacity = 1

choixCouleurEstOuvert = true

//Assigner les gestionnaires d'événements aux boutons

document.querySelector('.red').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

couleurChoisie('rgb(255, 6, 0)')

})

document.querySelector('.green').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

couleurChoisie('rgb(0, 170, 69)')

})

document.querySelector('.blue').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

couleurChoisie('rgb(0, 150, 224)')

})

document.querySelector('.yellow').addEventListener('click', (e) => {

// passe le nom de la classe pour la couleur

couleurChoisie('rgb(255, 222, 0)')

})

}

const couleurChoisie = (rgb) => {

//attribuer la couleur au joker qui se trouve sur le dessus de la pile de jeu

couleurBoutton.play()

jouerPile[jouerPile.length - 1].color = rgb

// cache le choix de couleur

cacheChoixCouleur()

tourJoueur = false;

setTimeout(jouerOrdi, ordiDelai)}

function cacheChoixCouleur() {

const choixCouleur = document.querySelector('.color-picker')

choixCouleur.style.opacity = 0

choixCouleurEstOuvert = false

}

const passOuInversion = () => {

// vérifie si c'est changement de tour ou passe son tour

if (jouerPile[jouerPile.length - 1].changeTour) {

tourJoueur = false

// au tour de l'ordi

setTimeout(jouerOrdi, ordiDelai)

}

}

// met à jour les noms des joueurs avec le nom de celui dont c'est le tour

//mise en subrillance

const afficheAuTourDom = () => {

if (tourJoueur) {

document.querySelector('.player-score-title').style.color = 'rgb(100, 150, 150)'

document.querySelector('.cpu-score-title').style.color = 'rgb(6, 37, 62)'

}

else {

document.querySelector('.player-score-title').style.color = 'rgb(6, 37, 62)'

document.querySelector('.cpu-score-title').style.color = 'rgb(100, 150, 150)'

}

}

//Fin de la partie et du jeu

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

const compterPoint = (perdantMain) => {

//Initialisation à 0

let points = 0

for (const card of perdantMain) {

points += card.points

}

if (perdantMain == ordiMain) {

ordiScore += points

}

else {

joueurScore += points

}

}

const majScores = () => {

// met à jour le score de l'ordinateur

ordiScoreDom.innerHTML = ordiScore

// Si la gagne est pour l'ordi

if (ordiScore < gameOver / 2) ordiScoreDom.style.color = 'rgb(0, 140, 0)' // couleur initiale du score en vert

else ordiScoreDom.style.color = 'rgb(121, 2, 2)' // évolution de la couleur en rouge

// met à jour le score du joueur

joueurScoreDom.innerHTML = joueurScore

// Si la gagne est pour le joueur

if (joueurScore < gameOver / 2) joueurScoreDom.style.color = 'rgb(0, 140, 0)'

else joueurScoreDom.style.color = 'rgb(121, 2, 2)'

}

const verifiePourGagnant = () => {

// vérifie si personne n'a perdu

if (joueurScore < gameOver && ordiScore < gameOver) {

// si joueur a gagné

if (joueurMain.length === 0) {

gagnePartie.play()

finPartie(joueurMain)

}

// Si ordi a gagné

if (ordiMain.length === 0) {

partiePerdue.play()

finPartie(ordiMain)

}

}

// sinon fin du jeu car la personne a atteint le score limite !

else {

// game over

finJeu()

}

}

const afficheOrdiCartes = () => {

if (ordiMain.length >= 1) {

ordiMainDom.innerHTML = ''

for (let i = 0; i < ordiMain.length; i++) {

// retourne les cartes

const ordiCarte = document.createElement('img')

ordiCarte.setAttribute('src', ordiMain[i].src)

ordiCarte.setAttribute('class', 'cpu')

ordiMainDom.appendChild(ordiCarte)

}

}

}

const cacheOrdiCartes = () => {

if (ordiMain.length >= 1) {

ordiMainDom.innerHTML = ''

for (let i = 0; i < ordiMain.length; i++) {

// retourne les cartes, vue de dos

const ordiCarte = document.createElement('img')

ordiCarte.setAttribute('src', 'images/back.png')

ordiCarte.setAttribute('class', 'cpu')

ordiMainDom.appendChild(ordiCarte)

}

}

}

const finPartie = (gagnant) => {

gameOn = false;

tourJoueur = !tourJoueur

if (ordiMain.length > 0) afficheOrdiCartes()

const finPartieDom = document.querySelector('.end-of-round')

const partieDom = document.querySelector('.round')

// affiche l'élément de fin de tour en fonction du gagnant

finPartieDom.classList.remove('hidden')

if (gagnant === joueurMain) partieDom.textContent = 'Tu as gagné la partie !'

else partieDom.textContent = 'Lordi a gagné la partie...'

// cache l'élements de fin de partie après 2 sec

setTimeout(() => {

finPartieDom.classList.add('hidden')

tourJoueur = !tourJoueur

nouvelleMain()

if (!tourJoueur) setTimeout(jouerOrdi, ordiDelai)

}, 3000)

}

const finJeu = () => {

gameOn = false;

if (ordiMain.length > 0) afficheOrdiCartes()

const finJeuDom = document.querySelector('.end-of-game')

const jeuDom = document.querySelector('.game')

// affiche l'élément de fin de partie et le format en fonction du gagnant

finJeuDom.classList.remove('hidden')

// Affichage des bulles

if (joueurScore > gameOver) {

partiePerdue.play()

jeuDom.textContent = 'Lordi a gagné le jeu... PEncore une partie?'

}

else {

gagneJeu.play()

jeuDom.textContent = 'Tu as gagné le jeu ! On recommence ?'

}

// ajoute un listener pour le bouton 'play again'

document.querySelector('.play-again').addEventListener('click', () => {

jouerEncore.play()

// cache l'élément de fin de jeu au clic

finJeuDom.classList.add('hidden')

joueurScore = 0

ordiScore = 0

majScores()

tourJoueur = !tourJoueur

nouvelleMain()

if (!tourJoueur) setTimeout(jouerOrdi, ordiDelai)

})

}

//Logique ordinateur comme une ia

const laisseOrdiTireCarte = () => {

if (jouerPile[jouerPile.length - 1].drawValue > 0) {

// ajoute le nombre de cartes basé sur la valeur de tirage de la dernière carte jouée.

for (let i = 0; i < jouerPile[jouerPile.length - 1].drawValue; i++) {

tireCarte(ordiMain)

}

}

}

// Choix de jeux de l'ordi

const jouerOrdi = () => {

if (!tourJoueur && gameOn) {

// En se basant sur la dernière carte jouée et ses propriétés, lordi va parcourir

// en boucle son tableau ordiMain, et toute carte qui correspond à la valeur ou à

// la couleur de la dernière carte jouée sera placée dans le tableau playableCartes

// avec les jokers que lordi peut détenir.

// Création d'un tableau temporaire playable de cartes jouables basé sur la

//dernière carte jouée

const playable = deterrmineCartesPlayable()

// Si pas de cartes à jouer

if (playable.length === 0) {

// pioche carte

tireCarte(ordiMain)

// fin du tour

setTimeout(() => {

// L'ordi a finit son tour, c'est maintenant au tour du joueur

tourJoueur = true

return

}, 500)

}

//Si une carte jouable (présente dans le tableau "playable", il joue la carte)

else if (playable.length === 1) {

setTimeout(joueOrdiCarte, 300, playable[0])

}

// Si il y a plus d'1 carte jouable

else if (playable.length > 1) {

// application de la stratégie définie précédemment

let carteChoisie = miseEnPlaceStrategie(playable)

setTimeout(joueOrdiCarte, 300, carteChoisie)

}

}

//Fonctions spécifiques à l'ordinateur

// Définition tableau "playable" cartes

function deterrmineCartesPlayable() {

const playableCartes = []

console.log(jouerPile[jouerPile.length - 1])

for (let i = 0; i < ordiMain.length; i++) {

// si même couleur ou même chiffre

if (ordiMain[i].color === jouerPile[jouerPile.length - 1].color || ordiMain[i].valeur === jouerPile[jouerPile.length - 1].valeur || ordiMain[i].color === 'any' || jouerPile[jouerPile.length - 1].color === 'any') {

let validCard = ordiMain.splice(i, 1)

// Le tableau se remplit avec les cartes qui correspondent à ce critère

playableCartes.push(validCard[0])

}

}

console.log('playable cards:')

return playableCartes

}

function miseEnPlaceStrategie(playable) {

let carteIndex

// Détermination de la stratégie de manière aléatoire à chaque tour

// Si la variable aléatoire est supérieure à 0,5, l'ordi donnera la priorité aux cartes

// d'action (jockers) 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.

// Initialisation aléatoire de la stratégie

let strategie = Math.random()

// Si la variable aléatoire est supérieure à 0,5

if (jouerPile.length > 2 && (strategie > 0.7 || joueurMain.length < 3 || ordiMain.length > (joueurMain.length \* 2) || (jouerPile[jouerPile.length - 1].joueParjoueur === true && jouerPile[jouerPile.length - 1].drawValue > 0) || (jouerPile[jouerPile.length - 2].joueParjoueur === true && jouerPile[jouerPile.length - 1].drawValue > 0))) {

// Lordi choisis et priorise la carte chiffre avec le nombre de points le plus élevé

let plusHauteValeur = 0

for (let i = 0; i < playable.length; i++){

if (playable[i].valeur > plusHauteValeur) {

plusHauteValeur = playable[i].valeur

carteIndex = i

}

}

// joue la carte déterminée par stratégie

// retire la carte depuis playable

carteChoisie = playable.splice(carteIndex, 1)

// retourne playable de ordiMain

returnPlayableAMain()

}

// Si la variable aléatoire est inférieure à 0,5

else {

// sinon priorise les cartes jokers comme la couleur ou le nombre de cartes

let plusFaibleValeur = 14

for (let i = 0; i < playable.length; i++){

if (playable[i].valeur < plusFaibleValeur) {

plusFaibleValeur = playable[i].valeur

carteIndex = i

}

}

// joue la carte déterminée par stratégie

// retire la carte depuis playable

carteChoisie = playable.splice(carteIndex, 1)

returnPlayableAMain()

}

return carteChoisie[0]

function returnPlayableAMain() {

if (playable.length > 0) {

for (const card of playable) {

ordiMain.push(card)

}

}

}

}

function joueOrdiCarte(carteChoisie) {

//carte aléatoire de ordiMainDom

const ordiDomCartes = ordiMainDom.childNodes

ordiDomCartes[Math.floor(Math.random() \* ordiDomCartes.length)].classList.add('cpu-play-card')

SonJouerCarte()

// debugger

setTimeout(() => {

jouerPile.push(carteChoisie)

// maj jeuPileDom

majJouerPileDom()

// vérifie si lordi a joué au hasard

if (jouerPile[jouerPile.length - 1].color === 'any' && jouerPile[jouerPile.length - 1].drawValue === 0 && jouerPile[jouerPile.length - 1].joueParjoueur === false) {

chooseColorAfterWild()

}

// vérifie la taille de la main de l'ordi et maj ordiMainDom

if (ordiMain.length >= 1) {

majMain(ordiMain)

if (ordiMain.length === 1) {

afficheUno(ordiUno)

}

}

// si fin du tour

else {

majMain(ordiMain)

setTimeout(() => {

compterPoint(joueurMain)

majScores()

verifiePourGagnant()

}, 1200)

}

// si lordi tire une carte

if (carteChoisie.drawValue > 0) {

PenalitéPlusQuatreCarte()

setTimeout(() => {

for (let i = 0; i < carteChoisie.drawValue; i++) {

tireCarte(joueurMain)

}

checkChangeTurn()

},1000)

}

// sinon

else setTimeout(checkChangeTurn, 500)

}, 500)

function checkChangeTurn() {

if (carteChoisie.changeTour) {

// si changement de tour, tourJoueur = true

tourJoueur = true

return

}

else {

// pas de changment, au tour de lordi

setTimeout(jouerOrdi, ordiDelai)

}

}

}

// FOnction choix couleur après carte wild

function chooseColorAfterWild() {

// 4 couleurs possibles

const colors = ['rgb(255, 6, 0)', 'rgb(0, 170, 69)', 'rgb(0, 150, 224)', 'rgb(255, 222, 0)']

const colorsInHand = [0, 0, 0, 0]

// Le processeur vérifie combien il a de cartes de chaque couleur.

for (const card of ordiMain) {

// itérations pour chaque couleur en comparant les deux listes

if (card.color === colors[0]) colorsInHand[0]++

if (card.color === colors[1]) colorsInHand[1]++

if (card.color === colors[2]) colorsInHand[2]++

if (card.color === colors[3]) colorsInHand[3]++

}

// trouver l'indice de la valeur maximale

let indexOfMax = colorsInHand.indexOf(Math.max(...colorsInHand))

// carte wild et choix des couleurs

const wildCardDom = jouerPileDom.childNodes[0]

wildCardDom.style.border = '5px solid ' + colors[indexOfMax]

wildCardDom.style.width = '105px'

jouerPile[jouerPile.length - 1].color = colors[indexOfMax]

}

}

// Wild +4

const PenalitéPlusQuatreCarte = () => {

plusCarteFX.play()

jouerPileDom.classList.add('shout')

setTimeout(() => {

jouerPileDom.classList.remove('shout')

}, 1000)

}

const playPlayerCard = (index) => {

let cardToPlay = joueurMain.splice(index, 1)

cardToPlay[0].joueParjoueur = true

jouerPile.push(cardToPlay[0])

// vide la pile de jeux

majJouerPileDom()

}

///////Principales fonctions du jeu - MAIN ////////

// Jeu

const startGame = () => {

if (!preLoaded) {

preLoadImgs()

preLoaded = true

}

// initialisation des scores

joueurScore = 0

ordiScore = 0

listenForDevMode()

setInterval(afficheAuTourDom, 100)

nouvelleMain()

majScores()

if (!tourJoueur) setTimeout(jouerOrdi, ordiDelai)

// Listeners sur joueurMainDom et piochePileDom au clic

// joueurMainDom

joueurMainDom.addEventListener('click', (event) => {

if (tourJoueur && !choixCouleurEstOuvert && event.target.getAttribute('id')) {

const lastCardDom = jouerPileDom.childNodes[0]

if (lastCardDom.style !== '100px') {

lastCardDom.style.width = '100px'

lastCardDom.style.border = 'none'

}

// utiliser l'ID de la cible pour trouver l'objet carte dans le tableau

let index = parseInt(event.target.getAttribute('id'))

// si la valeur ou la couleur correspondent à la carte de la pile

if (joueurMain[index].valeur === jouerPile[jouerPile.length - 1].valeur || joueurMain[index].color === jouerPile[jouerPile.length - 1].color || joueurMain[index].color === 'any' || jouerPile[jouerPile.length - 1].color === 'any') {

// carte animée au clic

SonJouerCarte()

event.target.classList.add('play-card')

console.log('you played', event.target) // TODO: remove

setTimeout(() => {

playPlayerCard(index)

// appelle cpuTurn pour ajouter des cartes s'il y en a à ajouter

laisseOrdiTireCarte()

// vérifié la taille de joueurMain et maj DOM

if (joueurMain.length >= 1) {

majMain(joueurMain)

// Si plus qu'une carte dans sa main --> Uno

if (joueurMain.length === 1) afficheUno(joueurUno)

}

// FIn de la partie --> maj points et scores et on définit le gagnant

else {

majMain(joueurMain)

setTimeout(() => {

compterPoint(ordiMain)

majScores()

verifiePourGagnant()

}, 1200)

}

//vérifie si carte tirée et carte "wild" (choix des couleurs)

if (jouerPile[jouerPile.length - 1].color === 'any' && jouerPile[jouerPile.length - 1].drawValue === 0 && jouerPile[jouerPile.length - 1].joueParjoueur) {

// nouvelle couleur

afficheChoixCouleur()

return

}

passOuInversion();

}, 1000)

}

}

})

let areYouSure = false

piochePileDom.addEventListener('click', () => {

if (tourJoueur && !choixCouleurEstOuvert) {

tireCarte(joueurMain)

setTimeout(() => {

tourJoueur = false;

setTimeout(jouerOrdi, ordiDelai)

}, 500)

}

})

}

let ordiVisible = false

const listenForDevMode = () => {

document.addEventListener('keydown', event => {

const key = event.key.toLowerCase()

console.log(key)

})

}

startGame()