Documentation complète du jeu de Blackjack en JavaScript

Table des matières

Introduction

Cette application est une implémentation complète du jeu de cartes Blackjack (ou "21") en JavaScript pur. Le jeu permet à un utilisateur de jouer contre un croupier virtuel en suivant les règles classiques du Blackjack. Développée en HTML, CSS et JavaScript, cette application offre une interface utilisateur attrayante avec un design moderne et immersif, tout en proposant l'ensemble des fonctionnalités essentielles du jeu.

Caractéristiques principales :

- Interface utilisateur immersive avec effets visuels et animations
- Représentation visuelle des cartes avec symboles et couleurs appropriés
- Logique complète du jeu de Blackjack, incluant la gestion des As (valeur 1 ou 11)
- Fonctionnalités de tirage de carte ("hit"), de maintien de la main ("stand") et de réinitialisation
- Affichage des valeurs des mains et du résultat de la partie
- Design responsive adapté à tous les appareils

Structure du projet

L'application est contenue dans un seul fichier HTML qui intègre :

- Le balisage HTML pour la structure du jeu

- Les styles CSS intégrés dans une balise `<style>`
- Le code JavaScript intégré dans une balise `<script>`

Cette approche "tout-en-un" facilite le déploiement et l'utilisation de l'application sans dépendances externes.

Interface utilisateur

L'interface utilisateur est composée des éléments suivants :

1. **En-tête** : Titre principal "Blackjack" avec effet de texte
2. **Zone de jeu** : Un conteneur principal divisé en deux sections pour afficher :
    - La main du joueur avec ses cartes
    - La main du croupier avec ses cartes (dont une face cachée initialement)
3. **Zone de résultat** : Un espace dédié à l'affichage du résultat de la partie
4. **Boutons de contrôle** :
    - "Tirer" : pour demander une carte supplémentaire
    - "Rester" : pour garder sa main actuelle et laisser jouer le croupier
    - "Recommencer" : pour initialiser une nouvelle partie

Styles CSS

Design global et arrière-plan

L'application utilise un design sombre et immersif avec un arrière-plan animé :

```
body {
    background: linear-gradient(135deg, #0b132b, #1c2541);
    background-image: url('https://i.postimg.cc/9fVBTmvD/background.gif');
    background-size: cover;
    background-position: center;
    background-attachment: fixed;
}

body::before {
    content: '';
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(11, 27, 51, 0.7);
    z-index: -1;
}
```

Un overlay semi-transparent est ajouté pour améliorer la lisibilité du contenu sur l'arrière-plan animé.

Conteneur principal et sections de jeu

Le jeu est présenté dans un conteneur avec effet de verre (glassmorphism) :

```
#game {
    background: rgba(28, 37, 65, 0.8);
    padding: 30px;
    width: 90%;
    max-width: 800px;
    border-radius: 20px;
    backdrop-filter: blur(10px);
    border: 1px solid rgba(255, 255, 255, 0.1);
    box-shadow: 0 8px 32px rgba(0, 0, 0, 0.3);
}
```

Les sections pour les mains du joueur et du croupier sont stylisées de manière uniforme :

```
.cards-section {
    text-align: center;
    flex: 1;
    background: rgba(58, 80, 107, 0.3);
    padding: 20px;
    border-radius: 15px;
    border: 1px solid rgba(255, 255, 255, 0.1);
}
```

Représentation visuelle des cartes

Les cartes sont stylisées pour ressembler à de vraies cartes à jouer :

```
.card {
    width: 70px;
    height: 100px;
    background: white;
    border-radius: 8px;
    display: flex;
    flex-direction: column;
    justify-content: space-between;
    padding: 5px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.2);
    position: relative;
    transition: transform 0.3s ease;
}

.card.red {
    color: #d00000;
}

.card.black {
    color: #000;
}
```

Les cartes rouges (Cœurs et Carreaux) et noires (Trèfles et Piques) sont différenciées par leur couleur, et un effet de survol élève légèrement les cartes :

```
.card:hover {
    transform: translateY(-5px);
}
```

Carte face cachée

Une carte face cachée est représentée avec un motif spécifique :

```
.card.hidden {
    background: linear-gradient(45deg, #b30000 25%, transparent 25%),
                linear-gradient(-45deg, #b30000 25%, transparent 25%),
                linear-gradient(45deg, transparent 75%, #b30000 75%),
                linear-gradient(-45deg, transparent 75%, #b30000 75%);
    background-size: 20px 20px;
    background-color: #d00000;
    border: 2px solid #fff;
}
```

Boutons et feedback visuel

Les boutons sont stylisés avec des dégradés de couleur et des effets de survol :

```
button {
    padding: 12px 25px;
    font-size: 1.1rem;
    font-weight: 600;
    border: none;
    border-radius: 25px;
    cursor: pointer;
    transition: all 0.3s ease;
    text-transform: uppercase;
    letter-spacing: 1px;
    min-width: 150px;
}

button:hover {
    transform: translateY(-3px);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}
```

Chaque bouton a une couleur spécifique correspondant à son action :

- "Tirer" : vert (action de progression)
- "Rester" : orange (action de finalisation)
- "Recommencer" : rouge (action de réinitialisation)

Feedback des résultats

La zone de résultat change de couleur selon l'issue de la partie :

```
#result.victory {
    background: linear-gradient(135deg, #4CAF50, #45a049);
    border-color: #4CAF50;
}

#result.defeat {
    background: linear-gradient(135deg, #d00000, #b30000);
    border-color: #d00000;
}
```

Design responsive

L'application s'adapte aux écrans de taille réduite :

```
@media (max-width: 768px) {
    h1 {
        font-size: 2rem;
    }

    .cards-container {
        flex-direction: column;
    }

    .button-container {
        flex-direction: column;
    }

    button {
        width: 100%;
    }
}
```

Fonctionnalités JavaScript

Gestion du jeu de cartes

Création du paquet

La fonction `creerPaquet()` génère un jeu complet de 52 cartes :

```
function creerPaquet() {
    const couleurs = ['Cœurs', 'Carreaux', 'Trèfles', 'Piques'];
    const valeurs = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'V', 'D', 'R', 'A'];
    const paquet = [];
    for (let couleur of couleurs) {
        for (let valeur of valeurs) {
            paquet.push({ valeur: valeur, couleur: couleur });
        }
    }
    return paquet;
}
```

Mélange du paquet

La fonction `melangerPaquet()` utilise l'algorithme de Fisher-Yates pour mélanger aléatoirement les cartes :

```
function melangerPaquet(paquet) {
    for (let i = paquet.length - 1; i > 0; i--) {
        const j = Math.floor(Math.random() * (i + 1));
        [paquet[i], paquet[j]] = [paquet[j], paquet[i]];
    }
    return paquet;
}
```

Calcul des valeurs des mains

La fonction `calculerValeurMain()` détermine la valeur d'une main en tenant compte des règles spécifiques pour les As :

```
function calculerValeurMain(main) {
    let valeur = 0, nombreAs = 0;
    for (let carte of main) {
        if (carte.valeur === 'A') {
            valeur += 11;
            nombreAs++;
        } else if (['R', 'D', 'V'].includes(carte.valeur)) {
            valeur += 10;
        } else {
            valeur += parseInt(carte.valeur);
        }
    }
    while (valeur > 21 && nombreAs > 0) {
        valeur -= 10;
        nombreAs--;
    }
    return valeur;
}
```

Cette fonction gère intelligemment les As qui peuvent valoir 1 ou 11 points selon la situation, pour optimiser la valeur de la main sans dépasser 21.

Affichage des cartes

Symboles des couleurs

La fonction `getSuitSymbol()` convertit le nom de la couleur en son symbole correspondant :

```
function getSuitSymbol(couleur) {
    const symbols = {
        'Cœurs': '♥',
        'Carreaux': '♦',
        'Trèfles': '♣',
        'Piques': '♠'
    };
    return symbols[couleur];
}
```

Détermination de la couleur

La fonction `isRedSuit()` détermine si une couleur est rouge (Cœurs ou Carreaux) :

```
function isRedSuit(couleur) {
    return couleur === 'Cœurs' || couleur === 'Carreaux';
}
```

Création des éléments de carte

Les fonctions `createCardElement()` et `createHiddenCard()` génèrent les éléments HTML représentant les cartes :

```
function createCardElement(carte) {
    const cardDiv = document.createElement('div');
    cardDiv.className = `card ${isRedSuit(carte.couleur) ? 'red' : 'black'}`;

    cardDiv.innerHTML = `
        <div class="card-value">${carte.valeur}</div>
        <div class="card-suit">${getSuitSymbol(carte.couleur)}</div>
        <div class="card-value-bottom">${carte.valeur}</div>
    `;

    return cardDiv;
}

function createHiddenCard() {
    const cardDiv = document.createElement('div');
    cardDiv.className = 'card hidden';
    return cardDiv;
}
```

Affichage des mains

La fonction `afficherMains()` met à jour l'interface pour afficher les cartes du joueur et du croupier :

```
function afficherMains(devoilerCarteCroupier = false) {
    const playerCardsDiv = document.getElementById('player-cards');
    const dealerCardsDiv = document.getElementById('dealer-cards');

    // Code d'affichage des cartes et des valeurs...
}
```

Cette fonction prend un paramètre `devoilerCarteCroupier` qui détermine si la deuxième carte du croupier doit être visible ou non.

Logique du jeu

Initialisation du jeu

La fonction `initJeu()` prépare une nouvelle partie :

```
function initJeu() {
    paquet = melangerPaquet(creerPaquet());
    mainJoueur = [paquet.pop(), paquet.pop()];
    mainCroupier = [paquet.pop(), paquet.pop()];
    afficherMains();
    document.getElementById('hitBtn').disabled = false;
    document.getElementById('standBtn').disabled = false;
    afficherResultat('', ''); // Réinitialise le résultat
}
```

Gestion des actions du joueur

Action "Tirer"

L'événement de clic sur le bouton "Tirer" ajoute une carte à la main du joueur et vérifie si le joueur dépasse 21 :

```
document.getElementById('hitBtn').addEventListener('click', () => {
    if (calculerValeurMain(mainJoueur) < 21) {
        mainJoueur.push(paquet.pop());
        afficherMains();

        if (calculerValeurMain(mainJoueur) > 21) {
            afficherResultat("Vous avez dépassé 21 ! Vous perdez !", 'defeat');
            desactiverBoutons();
        }
    }
});
```

Action "Rester"

L'événement de clic sur le bouton "Rester" finalise la main du joueur et fait jouer le croupier selon les règles standard (tirer jusqu'à 17 ou plus) :

```
document.getElementById('standBtn').addEventListener('click', () => {
    desactiverBoutons();
    while (calculerValeurMain(mainCroupier) < 17) {
        mainCroupier.push(paquet.pop());
    }
    afficherMains(true);

    const valeurJoueur = calculerValeurMain(mainJoueur);
    const valeurCroupier = calculerValeurMain(mainCroupier);

    if (valeurCroupier > 21 || valeurJoueur > valeurCroupier) {
        afficherResultat("Vous gagnez !", 'victory');
    } else if (valeurCroupier > valeurJoueur) {
        afficherResultat("Le croupier gagne !", 'defeat');
    } else {
        afficherResultat("Égalité !", '');
    }
});
```

Réinitialisation

L'événement de clic sur le bouton "Recommencer" lance une nouvelle partie :

```
document.getElementById('resetBtn').addEventListener('click', initJeu);
```

Fonctions utilitaires

Désactivation des boutons

La fonction `desactiverBoutons()` empêche le joueur d'effectuer des actions une fois la partie terminée :

```
function desactiverBoutons() {
    document.getElementById('hitBtn').disabled = true;
    document.getElementById('standBtn').disabled = true;
}
```

Affichage des résultats

La fonction `afficherResultat()` met à jour le message de résultat avec une classe CSS appropriée pour le style :

```
function afficherResultat(message, type) {
    const resultDiv = document.getElementById('result');
    resultDiv.innerText = message;
    resultDiv.classList.remove('victory', 'defeat');
    if (type === 'victory') resultDiv.classList.add('victory');
    if (type === 'defeat') resultDiv.classList.add('defeat');
}
```

Guide d'utilisation

Déroulement d'une partie

1. **Début de partie** :
    - Le jeu démarre automatiquement au chargement de la page
    - Le joueur et le croupier reçoivent chacun deux cartes
    - Une seule carte du croupier est visible
2. **Tour du joueur** :
    - Le joueur peut choisir de "Tirer" pour recevoir une nouvelle carte
    - Si la valeur de sa main dépasse 21, il perd immédiatement (bust)
    - Le joueur peut choisir de "Rester" pour conserver sa main actuelle et passer au tour du croupier
3. **Tour du croupier** :
    - Le croupier révèle sa deuxième carte
    - Il tire automatiquement des cartes jusqu'à ce que sa main atteigne au moins 17 points
4. **Fin de partie** :
    - Si le croupier dépasse 21, le joueur gagne
    - Si le joueur a une main de valeur supérieure au croupier, il gagne
    - Si le croupier a une main de valeur supérieure au joueur, le joueur perd
    - Si les deux mains ont la même valeur, c'est une égalité
5. **Nouvelle partie** :
    - Le joueur peut cliquer sur "Recommencer" pour initialiser une nouvelle partie

Règles spécifiques

- Les cartes numériques (2-10) valent leur valeur faciale
- Les figures (Valet, Dame, Roi) valent 10 points
- L'As vaut 11 points, mais passe automatiquement à 1 point si nécessaire pour éviter de dépasser 21

Personnalisation

Modification des couleurs et du style

Pour personnaliser l'apparence du jeu, vous pouvez modifier les dégradés de couleur et les variables CSS dans la section de style :

```
body {
    background: linear-gradient(135deg, #0b132b, #1c2541); /* Dégradé de fond */
}

h1 {
    color: #ff922b; /* Couleur du titre */
    text-shadow: 0 0 10px rgba(255, 146, 43, 0.5); /* Ombre du texte */
}

/* Couleurs des boutons */
button#hitBtn {
    background: linear-gradient(135deg, #4CAF50, #45a049); /* Vert */
}

button#standBtn {
    background: linear-gradient(135deg, #ff922b, #f0a500); /* Orange */
}

button#resetBtn {
    background: linear-gradient(135deg, #d00000, #b30000); /* Rouge */
}
```