Développement d’un JEU 2048 sous forme d’application web en HTML / CSS / JavaScript.

Préparation: créez un compte sur scrimba.com ; familiarisez-vous avec le site web en créant un screencast - un scrim - (vous devez avoir un microphone et un casque audio fonctionnels). Montrez le screencast - scrim - pour prouver que votre installation est fonctionnelle.

Trouvez des sites internet ayant des ressources pédagogiques en français ou en anglais portant sur les thèmes HTML CSS et JavaScript. Quelques exemples:

|  |
| --- |
| Openclassrooms  <https://developer.mozilla.org/fr/>  developpez.net develpoppez.com pierre-giraud.com  w3schools fun-mooc learnjs.org cssdebutant.com html.com codecombat Codecademy stackoverflow (questions réponses) |

Votre but est de développer le jeu 2048 avec des technologies web : HTML/CSS/JavaScript.

Défi 1: EN HTML uniquement: afficher un tableau 4 fois 4 (sans traits ou bordures) avec des étoiles dans chaque case.

Test 1 : ce test est visuel, il doit y avoir un tableau affiché dans le navigateur et il est possible de voir la taille 4 \* 4 (par exemple avec des lignes) avec des étoiles dans chacune des cases.

Défi 2: En HTML et CSS: afficher un titre en utilisant la balise h1 au dessus du tableau, faites en sorte que le titre soit de couleur rouge. Vous devez utiliser le CSS pour configurer la couleur rouge, pas l’HTML, donc vous devez créer un fichier CSS en plus de votre fichier HTML.

Test 2 : ce test est visuel, structurel (structure des fichiers), et fonctionnel (vérification des erreurs dans la console de débogage du navigateur web). Pour le test visuel, un titre plus gros en rouge doit apparaître. Pour le test structurel, vérifier qu’il y a au moins deux fichiers : un de type HTML et un de type CSS, vérifier que les deux doivent être liés (c’est à dire que le HTML fait référence au CSS - avec le nom du fichier CSS indiqué dans le fichier HTML). Pour le test fonctionnel, ouvrez la console de débogage du navigateur et vérifiez qu’aucune erreur n’apparaisse.

**Défi 3: En JavaScript : écrire une fonction nommée bonjour qui affiche “bonjour” dans la console de débogage du navigateur.** Aidez-vous du code fourni ci-dessous et complétez la partie //TODO1.

function bonjour (){

// TODO1

}

Test 3 : créez une fonction testBonjour() qui fait appel à la fonction bonjour() en complétant //TODO2. La fonction testBonjour() ne doit être appelée qu’une et une seule fois au tout début du fichier javascript. La fonction bonjour() ne doit être appelée qu’une et une seule fois dans la fonction testBonjour()

Vérifiez aussi que le fichier HTML est bien lié au fichier javascript (si rien de s’affiche c’est sûrement la source du problème).

Vérifiez qu’un et un seul « bonjour » s’affiche dans la console du navigateur.

function testBonjour (){

// TODO2

}

la structure globale de votre code doit donc ressembler à cela :

//programme principal :

testBonjour() ;

function bonjour (){

// TODO1

}

function testBonjour (){

// TODO2

}

Enregistrer un screencast avec scrimba qui explique le Défi 3 et le Test 3 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 4: Afficher des bordures noires autour des cases du tableau en CSS. Ne pas modifier l’HTML mais uniquement le CSS pour afficher ces bordures.

Test 4 : c’est un test visuel uniquement : vérifiez que le tableau apparaît bien avec des bordures noires visuellement.

Défi 5: Agrandir les cases du tableau avec une taille fixe en CSS (pas en HTML). Le tableau doit être confortable pour jouer. Essayez d’afficher des cases carrées (et non rectangulaires).

Test 5 : test visuel, structurel et fonctionnel : visuellement le tableau doit avoir des grandes cases carrées et fonctionnellement il ne doit pas y avoir d’erreurs dans la console, structurellement le fichier HTML ne doit pas être modifié par rapport au défi précédent.

Ajouter du contenu dans une case (par exemple \*\*\*\*\* au lieu de \*) et vérifiez que la case reste carrée.

Défi 6: Attention, défi difficile (enfin, pour les débutants :) - En JavaScript: écrire une fonction  haut()  qui affiche “haut” dans la console en modifiant //TODO3.   
Écrire une fonction keyTester(event) qui va recevoir en paramètre l’événement ‘event’ correspondant à l’appui sur une touche du clavier. Cette fonction va ensuite vérifier quelle touche du clavier a été appuyée grâce à la propriété ‘key’ de l’événement (ici appelé event, en paramètre de la fonction keyTester(event). En fonction de la touche appuyée, elle va appeler la bonne fonction (touche haut -> fonction haut() etc.) (à remplacer dans //TODO5).  
Modifiez le programme principal au niveau de //TODO4 pour qu’à chaque fois qu’on appuie sur une touche du clavier, la fonction keyTester est appelée.

De ce fait , quand la touche “haut” du clavier est appuyée, la fonction haut() est appelée par la fonction keyTester. Important : ne pas utiliser la balise “body” en HTML (ou même une autre balise) mais utilisez plutôt “document” en JavaScript directement.

Test 6 : appuyez trois fois sur la touche haut du clavier et vérifiez que « haut » s’affiche trois fois dans la console, sans erreur. le programme principal doit être modifié pour activer le mécanisme de détection de la touche au niveau du premier //TODO4 dans la structure de code donnée ci-dessous.

la structure globale de votre code doit maintenant ressembler à cela :

//programme principal :

testBonjour() ;

//TODO4

function bonjour (){

// TODO1

}

function testBonjour (){

// TODO2

}

function keyTester (event){

// TODO5

}

function haut (){

// TODO3

}

Défi 7: De même, créez les fonctions bas() droite() gauche() en modifiant //TODO8 //TODO9 //TODO10 qui affichent la direction dans la console à chaque fois qu’on appuie sur les touches correspondantes.

Test 7 : appuyez 2 fois sur chaque touche directionnelle et voir que chaque direction est affichée deux fois.

le programme principal doit être modifié pour activer les mécanismes de détection des touches directionnelles au niveau des //TODO4 //TODO5 //TODO6 //TODO7 dans la structure de code donnée ci-dessous.

la structure globale de votre code doit maintenant ressembler à cela :

//programme principal :

testBonjour() ;

//TODO4

//TODO5

//TODO6

//TODO7

function bonjour (){

// TODO1

}

function testBonjour (){

// TODO2

}

function haut (){

// TODO3

}

function bas (){

// TODO8

}

function gauche (){

// TODO9

}

function droite (){

// TODO10

}

Défi 8: En JavaScript, écrire une fonction changeTitre () qui change le texte du titre (celui affiché en rouge du défi 2). Cette fonction n’a pas de valeur retournée (c’est à dire pas de return). Modifiez le programme principale pour que quand la touche t est appuyée, la fonction changeTitre () est appelée.

Test 8 : Pour tester, affichez d’abord la page avec le titre original, puis appuyez sur la touche “t” et observez que le titre change avec un nouveau texte.

Défi 9: En JavaScript, écrire une fonction score() qui affiche dans la console “score : x” avec x qui vaut 0 au début et qui est incrémenté de 1 à chaque fois qu’une des 4 touches directionnelles est appuyée. La variable x doit être initialisée au début, dans le programme principal. Modifiez ensuite le programme principal pour qu’à chaque fois que la touche « s » est appuyée, la fonction score() est appelée.

Test 9 : au début, appuyez sur la touche s et observez dans la console « score : 0 » appuyez ensuite une fois sur une touche directionnelle puis une fois sur la touche s et observez dans la console « score : 1 » . Appuyez ensuite 4 fois sur les touches directionnelles dans cet ordre : haut bas gauche droit et appuyez sur la touche s et observez dans la console « score : 5 ». Attention, le score s’affiche uniquement quand la touche s est appuyée.

SCREENCAST Défi 9: enregistrer un screencast avec scrimba qui explique le Défi 9 et le Test 9 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 10 : En HTML, écrivez en dessous du tableau “score =” . Utilisez ensuite une balise de type DIV juste après le texte “score = ” qui affichera le score. Modifiez la fonction score pour qu’en plus d’afficher le score dans la console, elle modifie la balise DIV avec le score. Comme avant, la touche s va appeler la fonction score.

Test 10 : Au début, appuyez sur la touche s et observez non seulement dans la console « score : 0 » mais aussi dans le rendu de la page web.

appuyez ensuite une fois sur une touche directionnelle puis une fois sur la touche s et observez dans la console « score : 1 » ainsi que dans la page web.

Appuyez ensuite 4 fois sur les touches directionnelles dans cet ordre : haut bas gauche droit et appuyez sur la touche s et observez dans la console « score : 5 » ainsi que dans la page web. Attention, le score s’affiche uniquement quand la touche s est appuyée.

Défi 11 : En JavaScript, écrire une fonction getTable() qui retourne le tableau HTML (avec un return).

|  |
| --- |
| **Indice 11**: vous pouvez utiliser un identifiant pour le tableau en HTML (par exemple <table id="mytable">) puis en javascript vous récupérez le tableau avec  var t = document.getElementById('myTable'); |

Test 11 : écrire une fonction testGetTable() qui appelle la fonction getTable(), récupère la valeur retournée par cette fonction dans une variable nommée ‘table’ et affiche dans la console la valeur de cette variable ‘table’.

Dans le programme principal (en haut du fichier javascript), utilisez ensuite l’évévenement ‘load’ pour appeler la fonction testGetTable() quand la page est complètement chargée , tel qu’expliqué ici (code d’exemple juste en dessous) <https://developer.mozilla.org/fr/docs/Web/API/Window/load_event>

Cela permet d’attendre que le tableau soit totalement chargé avant de tester la fonction testGetTable(), autrement, le tableau ne serait probablement pas chargé et la fonction getTable() ne renverrait RIEN.

Pour cela vous remplacerez le //TODO par l’appel de la fonction testGetTable() dans le code ci-dessous :

|  |
| --- |
| window.onload = (event) => {  console.log('La page est complètement chargée');  //TODO  }; |

vous devriez avoir dans la console la première balise du tableau qui s’affiche, comme par exemple (si ce n’est pas le cas, voir juste en dessous)

<table id="mytable">

Un erreur classique avec scrimba est que le fichier index.js soit lié au fichier HTML avec index.pack.js comme nom : allez dans le HTML et modifiez index.pack.js en index.js

Défi 12 : En JavaScript, écrire une fonction getCell(i,j) qui retourne (avec un return) la cellule de coordonnées i, j du tableau. Cette fonction va d’abord appeler getTable() pour récupérer le tableau puis va utiliser la syntaxe rows[i].cells[i] pour récupérer la case de coordonnées i,j du tableau (voir l’indice ci-dessous).

Test 12 : Écrire une fonction testGetCell() qui va tester la fonction getCell. Pour cela elle va appeler la fonction getCell plusieurs fois avec des coordonnées différentes. Elle récupérera les retours des différents appels dans des variables et elle affichera le contenu de ces variables dans la console en utilisant .innerHTML (ce qui veut dire le contenu le case du tableau), voir <https://developer.mozilla.org/fr/docs/Web/API/Element/innerHTML> pour plus d’explications.

Par exemple :

|  |
| --- |
| var first = getCell(0,0)  var second = getCell(2,3)  var third = getCell(0,1)  console.log(first.innerHTML)  console.log(second.innerHTML)  console.log(third.innerHTML) |

modifiez ensuite le window.onload dans le programme principal (du défi précédent) pour qu’elle appelle la fonction testGetCell() une fois le document chargé entièrement

|  |
| --- |
| **Indice 12**: vous pouvez récupérer le tableau dans une variable table puis utiliser la syntaxe table.rows[i].cells[j] pour récupérer la case de coordonnées i, j |

vérifiez que le résultat de l’affichage est bien le contenu des cases en modifiant le contenu des cases dans l’HTML.

Vérifiez que vous avez bien des td et aucun th dans votre table en HTML car sinon vous aurez une erreur de type « undefined »

SCREENCAST Défi 12: enregistrer un screencast avec scrimba qui explique le Défi 12 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 13 : modifiez l’HTML pour avoir des nombres différents que vous choisissez au hasard dans les cases (au lieu des étoiles). En javascript, modifiez la fonction testGetCell pour qu’elle affiche dans la console les contenus des cases + 1 (c’est à dire incrémentée de 1) en utilisant la fonction parseInt qui transforme une chaine de caractère en entier <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/parseInt>

En effet, innerHTML renvoie une chaine de caractère qu’il faut transformer en entier pour pouvoir ajouter 1 .

Test 13 : vérifiez dans la console que c’est bien le contenu des cases incrémenté de 1 qui s’affiche.

Défi 14: En JavaScript, écrire une fonction getValue(i,j) qui appeler getCell(i,j) , va transformer la chaîne de caractère retournée par getCell en entier et va retourner l’entier.

Test 14 : Écrire ensuite une fonction testGetValue() qui va tester la fonction getValue avec les coordonnées des cases diagonales (0,0) (1,1) (2,2) (3,3) et afficher les trois retours de résultat dans la console. Modifiez le window.onload dans le programme principal pour qu’elle appelle la fonction testGetValue() une fois le document chargé entièrement

Défi 15: En JavaScript, écrire une fonction setValue(i,j,val) ne retourne rien mais qui va appeler getCell pour avoir la case de coordonnées i,j et qui va modifier le texte contenu dans cette case en mettant la valeur val.

Test 15 : Écrire une fonction testSetValue() qui va tester la fonction setValue avec les appels de setValue(0,0,1) setValue(1,1,2) setValue(2,2,3) setValue(3,3,4).

Modifiez le window.onload dans le programme principal pour qu’elle appelle la fonction testSetValue() une fois le document chargé entièrement

Remarquez que les trois cases en diagonale se modifient avec les valeurs 1 2 3 et 4 (juste après le chargement de la page)

SCREENCAST Défi 15: enregistrer un screencast avec scrimba qui explique le Défi 15 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 16: En JavaScript, écrire une fonction showRow(i) qui va afficher le texte de toutes les cases de la ligne i (Row en anglais veut dire ligne) dans la console. Pour cela pour utiliserez getValue(i,j) pour obtenir les valeurs des cases de la ligne. Vous utiliserez aussi une boucle for qui va parcourir toutes les cases de la ligne en question grâce à la variable j qui varie de 0 jusqu’au nombre de cases de la ligne - 1.

Test 16 : écrire une fonction testShowRow() qui va tester la fonction showRow avec showRow(0) showRow(1) showRow(2) et showRow(3)

Modifiez le window.onload dans le programme principal pour qu’elle appelle la fonction testShowRow() une fois le document chargé entièrement

Modifiez ensuite des éléments dans l’HTML pour vérifier que tout fonctionne correctement. Par exemple, changez la case 0,0 avec le texte “M” , la case 0,1 avec le texte “N”, la case 0,2 avec le texte “O” et la case 0,3 avec le texte “P” et regardez si showRow(0) affiche bien les valeurs “M” “N” “O” “P”.

Défi 17: En JavaScript, écrire une fonction showCol(j) qui va afficher le texte de toutes les cases de la colonne j. Pour cela pour utiliserez getValue(i,j) pour obtenir les valeurs des cases de la colonne . Vous utiliserez aussi une boucle for qui va parcourir toutes les cases de la colonne en question grâce à la variable i qui varie de 0 jusqu’au nombre de cases de la colonne - 1.

Test 17 : écrire une fonction testShowCol() qui va tester la fonction showCol avec showCol(0) showCol(1) showCol(2) et showCol(3).

Modifiez le window.onload dans le programme principal pour qu’elle appelle la fonction testShowRow() une fois le document chargé entièrement

Modifiez ensuite des éléments dans l’HTML pour vérifier que tout fonctionne correctement. Par exemple, changez la case 0,0 avec le texte “E” , la case 1,0 avec le texte “F”, la case 2,0 avec le texte “G” et la case 3,0 avec le texte “H” et regardez si showCol(0) affiche bien les valeurs “E” “F” “G” “H”.

Défi 18: En JavaScript, écrire une fonction setRow(i,a,b,c,d) qui va mettre, pour la ligne i, les quatre valeurs a b c d respectivement dans les cases ayant pour indice i,0 puis i,1 puis i,2 puis i,3 . Pour cela, vous utiliserez la fonction setValue(i,j,val).

Test 18 : Ecrire une fonction testSetRow() qui va tester cette fonction avec setRow(0,1,2,3,4) setRow(1,5,6,7,8) setRow(2,9,10,11,12) et setRow(3,13,14,15,16) et qui sera exécutée une fois le document chargé entièrement (avec la modification de window.onload)

SCREENCAST Défi 18: enregistrer un screencast avec scrimba qui explique le Défi 18 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 19: Écrire une fonction testInit() qui appelle la fonction init() définie ci-dessous.

|  |
| --- |
| function init(){  var tab = [  [1, 2, 3, 4],  [5, 6, 7, 8],  [9, 10, 11, 12],  [13, 14, 15, 16]  ];    for(let i=0;i<=3;i++){  for(let j=0;j<=3;j++){  setValue(i,j,tab[i][j]);  }  }  } |

Test 19 : faites en sorte que testInit () est exécutée à chaque fois qu’on appuie sur la touche “i” du clavier. Vérifiez que vous voyez en HTML les même valeurs que ceux de la matrice tab définie dans init() quand vous appuyez sur la touche i.

Défi 20: Écrire une fonction testGetRandomInt () qui va appeler 2 fois la fonction getRandomInt (ci-dessous) pour générer aléatoirement deux entiers entre 0 (inclus) et 3 (inclus). Ces deux entiers seront des indices i et j et cela permet de choisir de manière aléatoire un indice i et un indice j.

La fonction getRandomInt(min,max) ci-dessous renvoie justement un entier aléatoire entre min (inclus) et max (inclus). Utiliser ces deux entiers aléatoires (par exemple i,j) pour écrire le caractère @ dans la case i,j (en utilisant la fonction setValue(i,j,”@” ) ). Comme i et j sont aléatoire, le caractère @ va s’afficher dans une case aléatoire.

|  |
| --- |
| function getRandomInt(min, max) {  min = Math.ceil(min);  max = Math.floor(max);  return Math.floor(Math.random() \* (max - min + 1)) + min;  } |

test 20 : Faites en sorte que la fonction testGetRandomInt soit appellée à chaque fois qu’on appuie sur la touche « a » du clavier. Essayez ensuite d’appuyer plusieurs fois sur la touche « a » pour voir si le caractère @ est écrit dans des cases aléatoires.

Défi 21: Écrire une fonction getRandom2or4(*percent*) qui renvoie la valeur 2 dans *percent* % des cas et la valeur 4 dans (100 - *percent*)% des cas. Par exemple, si percent=80 cette fonction va renvoyer 2 dans 80% des cas et 4 dans 20% des cas.

|  |
| --- |
| indice 21: La fonction getRandom2or4(*percent*) va utiliser getRandomInt pour générer un entier aléatoire avec une borne min égale à 0 et une borne max égale à 100. Si l’entier est strictement inférieur à *percent* la fonction renvoie 2 sinon 4. |

Test 21 : écrire une fonction testGetRandom2or4() qui va tester 100 fois la fonction avec percent = 80 - c’est à dire que la fonction getRandom2or4(80) sera appelée 100 fois - récupérer la valeur de retour et afficher la valeur de retour dans la console. Il y aura donc 100 affichages du retour de l’appel de la fonction getRandom2or4(80) .

Faites en sorte que la fonction testGetRandomInt soit appellée à chaque fois qu’on appuie sur la touche « r » du clavier. Appuyez ensuite sur « r » et observez les 100 affichages dans la console. Est-ce qu’il y a « environ » 80 fois l’affichage de 2 et 20 fois l’affichage de 4 dans la console ? Si oui c’est que la fonction getRandom2or4(*percent*) doit probablement fonctionner.

SCREENCAST Défi 21: enregistrer un screencast avec scrimba qui explique le Défi 21 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 22: Écrire une fonction newGame() qui utilise getRandom2or4 et getRandomInt pour initialiser le jeu 2048. Pour cela, cette fonction doit choisir deux cases aléatoirement (réutilisez la méthode du défi 20). Tant que les deux cases sont égales, il faut choisir une nouvelle case jusqu’à ce que les deux cases soient différentes. Une fois que les deux cases sont différentes, il faut mettre une valeur 2 dans 85% des cas et une valeur 4 dans 15% des cas pour une case et la valeur 2 dans 86% des cas et la valeur 4 dans 14% des cas pour l’autre case.

Test 22 : Faites en sorte que la fonction newGame() soit déclenchée avec l’appui de la touche “n” . Appuyez ensuite plusieurs fois sur la touche « n » et observez que 2 cases différentes, choisies aléatoirement à chaque fois, ont des valeurs 2 dans 86% des cas et 4 dans 14% des cas (environ).

Défi 23: Modifiez d’abord la fonction init() pour initialiser les cases du jeu à vide (par exemple, une case vide contient le caractère ‘\*’:

|  |
| --- |
| function init(){  var tab = [  [‘\*’, ‘\*’, ‘\*’, ‘\*’],  [‘\*’, ‘\*’, ‘\*’, ‘\*’],  [‘\*’, ‘\*’, ‘\*’, ‘\*’],  [‘\*’, ‘\*’, ‘\*’, ‘\*’]  ];    for(let i=0;i<=3;i++){  for(let j=0;j<=3;j++){  setValue(i,j,tab[i][j]);  }  }  } |

Écrire ensuite une fonction isEmpty (i,j) qui permet de tester si une case est vide. Si la case est vide, la fonction renvoie true , sinon la fonction renvoie false . Cette fonction va utiliser la fonction getValue(i,j) puis tester si le texte retourné par getValue correspond au texte d’une case vide (par exemple ‘\*’). Elle va ensuite utiliser l’instruction return false; ou return true;

Voyez la documentation des booléens ici

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Boolean>

Test 23 : Testez cette fonction avec la fonction testIsEmpty() suivante.   
Faites en sorte que la fonction testIsEmpty soit appellée quand on appuie sur la touche « e ». Appuyez sur la touche e.

Toutes les cases doivent être détectées vides au départ.

|  |
| --- |
| function testIsEmpty(){  init();  for(let i=0;i<=3;i++){  for(let j=0;j<=3;j++){  if(isEmpty (i,j)){  console.log(“la case [“+i+”][“+j+”] est vide”);  }  }  }  } |
|  |

Modifiez ensuite avec la fonction testIsEmpty() (toujours appelée avec la touche e) pour que les fonctions init() et newGame() soit appelées. Appuyez sur « e » et vérifiez qu’Il y a deux cases qui ne doivent pas être vides. Vérifiez bien que le test fonctionne.

|  |
| --- |
| function testIsEmpty(){  init();  newGame();  for(let i=0;i<=3;i++){  for(let j=0;j<=3;j++){  if(isEmpty (i,j)){  console.log(“la case [“+i+”][“+j+”] est vide”);  }  }  }  } |
|  |

Défi 24: Ecrire la fonction moveRight(i) qui décale à droite les chiffres de la ligne i.

Test 24 : Ecrire une fonction testMoveRight(i) en changeant setRow avec les valeurs des 4 exemples ci-dessous et qui teste moveRight(i) avec ces 4 exemples :

Exemple 1:

si la ligne i est au départ

\* \* 2 \*

après l’appel de la fonction moveRight(i) elle sera:

\* \* \* 2

Exemple 2:

si la ligne i est au départ

4 \* 2 \*

après l’appel de la fonction moveRight(i) elle sera:

\* \* 4 2

Exemple 3:

si la ligne i est au départ

2 \* 2 2

après l’appel de la fonction moveRight(i) elle sera:

\* 2 2 2

Exemple 4:

si la ligne i est au départ

4 2 \* 4

après l’appel de la fonction moveRight(i) elle sera:

\* 4 2 4

Faites en sorte que la fonction testMoveRight() soit appelée à chaque fois qu’on appuie sur la touche flèche droite.

|  |
| --- |
| function testMoveRight(){  init();  setRow(0,’\*’,’\*’,’2’,’\*’);  moveRight(0);  } |
|  |

SCREENCAST Défi 24: enregistrer un screencast avec scrimba qui explique le Défi 24 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 25: Ecrivez la fonction moveLeft(i) pour avoir le même comportement que la fonction moveRight(i) précédemment écrite, mais cette fois dans le sens de déplacement gauche.

Test 25 : Testez de la même manière que pour moveRight(i).

Défi 26: Ecrivez la fonction moveUp(j) qui va faire la même chose que moveRight(i) et moveLeft(i) mais cette fois ci pour déplacer vers le haut la colonne j.

Test 26 : Testez avec la même approche que pour les fonctions moveRight et moveLeft, c’est à dire avec des exemples et une fonction de test. Vous pouvez, pour vous aider, créer une fonction setCol qui ressemble à setRow mais qui met des valeurs pour une colonne en particulier.

Défi 27: Ecrivez la fonction moveDown(j) qui va faire la même chose que moveRight(i) et moveLeft(i) mais cette fois ci pour déplacer vers le haut la colonne j.

Test 27 : Testez avec la même approche que pour les fonctions moveRight et moveLeft, c’est à dire avec des exemples et une fonction de test. Vous pouvez, pour vous aider, créer une fonction setCol qui ressemble à setRow mais qui met des valeurs pour une colonne en particulier.

Défi 28: Ecrivez la fonction fusionRight(i) qui fusionne deux cases qui ont le même chiffre côte à côte, en calculant l’addition. Attention, pour calculer l’addition, vous devez transformer le texte (format String) des cases en un Nombre (format entier) en utilisant Number(), par exemple, pour calculer l’addition de la case 2,3 avec la case 1,2:

var result = Number(getValue(2,3)) + Number(getValue(1,2));

En effet, getValue(i,j) ne renvoie que du texte qu’il faut transformer en nombre.

Test 28 : écrivez une fonction testFusionRight() qui fait en sorte de tester fusionRight avec ces exemples :

Exemple 1:

si la ligne i est au départ

\* \* 2 2

après l’appel de la fonction fusionRight(i) elle sera:

\* \* \* 4

Exemple 2:

si la ligne i est au départ

4 4 2 2

après l’appel de la fonction fusionRight(i) elle sera:

\* 8 \* 4

Exemple 3:

si la ligne i est au départ

\* 2 2 2

après l’appel de la fonction fusionRight(i) elle sera:

\* 2 \* 4

Exemple 4:

si la ligne i est au départ

\* 2 2 4

après l’appel de la fonction fusionRight(i) elle sera:

\* \* 4 4

Vous pouvez vous aider de cette fonction en changeant setRow avec les valeurs des 4 exemples ci-dessus.

Faites en sorte que cette fonction soit appelée quand on appuie sur la touche flèche droite (et donc ce ne sera plus la fonction testMoveRight qui sera appelée)

|  |
| --- |
| function testFusionRight(){  init();  setRow(0,’\*’,’\*’,’2’,’2’);  fusionRight(0);  } |
|  |

Défi 29: de la même manière que fusionRight(i), écrivez la fonction fusionLeft(i)

Test 29 : testez-la avec la même approche que précédemment. Faites attention à bien transformer le texte en nombre avec Number().

SCREENCAST Défi 29: enregistrer un screencast avec scrimba qui explique le Défi 29 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 30: Ecrivez les fonctions fusionUp(j) et fusionDown(j) pour fusionner cette fois les cases de la colonne j, avec la même approche que pour les fonctions fusionRight(i) et fusionLeft(i)

Test 30 : testez avec la même approche que précédemment

Défi 31: Ecrivez la fonction right(i) qui sera executée pour déplacer, fusionner puis re-déplacer une ligne vers la droite (algorithme appliqué à une ligne quand on appuie sur la touche droite pendant le jeu) :

|  |
| --- |
| function right(i){  moveRight(i);  fusionRight(i);  moveRight(i);  } |
|  |

Ecrivez ensuite la fonction rightall() qui sera executée pour toutes les lignes:

|  |
| --- |
| function rightall(){  for(let i=0;i<=3;i++){  right(i);  }  } |
|  |

Test 31 : faites en sorte que la fonction rightall soit appelée quand on appuie sur la flèche droite et vérifiez le comportement avec plusieurs exemples que vous inventerez (vous pouvez inventer des exemples extrêmes pour être sûr que toutes vos fonctions fonctionnent correctement).

Défi 32: En utilisant la même approche, écrivez les fonctions left(i), up(j), down(j) puis les fonctions leftall(), upall() et downall(). Faites en sorte que quand on appuie sur la touche droite, cela déclenche la fonction rightall(), sur la touche gauche la fonction leftall(), sur la touche haut la fonction upall() et sur la touche bas la fonction downall().

Test 32 Testez avec les exemples suivants:

Exemple 1:

si les valeurs des cases initiales sont:

\* \* 2 2   
4 2 \* \*  
4 4 \* \*  
2 \* \* 2

après l’appui de la touche droite, les valeurs doivent être:

\* \* \* 4   
\* \* 4 2  
\* \* \* 8  
\* \* \* 4

Exemple 2:

si les valeurs des cases initiales sont:

4 \* 2 \*   
\* 2 \* 4  
4 2 \* 4  
2 \* 2 2

après l’appui de la touche bas , les valeurs doivent être:

\* \* \* \*   
\* \* \* \*  
8 \* \* 8  
2 4 4 2

Défi 33: modifiez les fonctions moveRight(i) moveLeft(i) moveUp(j) moveDown(j) pour qu’elles détectent s’il y a eu au moins un mouvement, dans ce cas elles retournent la valeur 1, sinon elles retournent la valeur 0. Pour ce faire, vous pouvez utiliser une variable déclarée  
 var hasMoved = 0;  
 au début de la fonction. Dès qu’une case bouge dans l’algorithme de la fonction, vous changez la valeur de hasMoved à 1, avec l’instruction hasMoved = 1; placée aux bons endroits. Enfin, à la fin de la fonction, vous retournez la valeur de hasMoved avec l’instruction return hasMoved;

De la même manière, modifiez les fonctions fusionRight(i) fusionLeft(i) fusionUp(j) fusionDown(j) pour qu’elles détectent s’il y a eu au moins une fusion. De la même manière, elles retournent la valeur 1 s’il y a eu au moins une fusion, sinon elles retournent la valeur 0.

Test 33 : affichez la valeur de retour (celle renvoyée par return hasMoved ;) dans la console et vérifiez la cohérence par rapport au mouvements.

SCREENCAST Défi 33: enregistrer un screencast avec scrimba qui explique le Défi 33 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>

Défi 34: modifiez la fonction right(i) et rightall() pour qu’elles propagent la détection de mouvement ou de fusion, comme ceci:

|  |
| --- |
| function right(i){  var hasChanged = 0;  hasChanged = moveRight(i) + fusionRight(i) + moveRight(i);  if(hasChanged != 0) hasChanged = 1  return hasChanged;  } |
|  |

Modifiez ensuite la fonction rightall() comme ceci:

|  |
| --- |
| function rightall(){  var hasChanged = 0;  for(let i=0;i<=3;i++){  hasChanged = hasChanged + right(i);  }  if(hasChanged != 0) hasChanged = 1  return hasChanged;  } |
|  |

Si l’utilisateur appuie sur la touche droite et qu’aucun mouvement et aucune fusion n’a eu lieu, alors affichez un message dans la console “pas de mouvement et pas de fusion”.

Test 34 : Faites en sorte que quand l’utilisateur appuie sur la touche droite, cela exécute la fonction rightall() et récupère la valeur retournée par rightall(). Si la valeur retournée est 0 c’est qu’il n’y a pas eu de mouvement ni de fusion, il faut afficher le message, sinon il ne faut pas afficher le message.

Défi 35: Ecrivez une fonction hasEmpty() qui retourne 1 s’il y a au moins une case vide et qui retourne 0 s’il n’y a aucune case vide/ Pour cela, cette fonction utilise la fonction isEmpty(i,j) en parcourant toutes les cases pour vérifier si elles sont vides.

Test 35 : Testez avec une fonction testHasEmpty () qui va d’abord mettre des valeurs dans toutes les cases puis qui appelle hasEmpty() . Testez ensuite en mettant une case vide et toutes les autres non vides.

Défi 36: Ecrivez une fonction getEmpty() qui, s’il y a une case vide, et uniquement s’il y a une case vide, choisi une case au hasard et renvoie ses coordonnées i,j . Pour cela, on vérifie d’abord qu’il y a au moins une case vide avec la fonction hasEmpty(). S’il y a au moins une case vide, on utilise la fonction getRandomInt pour choisir i au hasard et pour choisir j au hasard. Tant que la case i,j n’est pas vide, il faut choisi un nouveau i et un nouveau j puis retenter. Quand on a trouvé une case vide, on retourne ses coordonnées avec l’instruction   
return [i,j];

Test 36 : écrire une fonction testGetEmpty() qui va utiliser la fonction getEmpty de cette manière pour récupérer les coordonées d’une case vide et afficher les coordonées. Faites en sorte que la fonction testGetEmpty() soit appelée quand la page est chargée (avec window.onload)

|  |
| --- |
| var coord = getEmpty();  var i = coord[0];  var j = coord[1]; |

Défi 37: Terminez le jeu ! A chaque fois qu’il y a au moins un mouvement ou une fusion, utilisez getEmpty pour choisir une case vide et y mettre la valeur 2 ou 4 avec une certaine probabilité. Vous devriez être capable de terminer le jeu vous même!

SCREENCAST Défi 37: enregistrer un screencast avec scrimba qui explique le Défi 37 en montrant ce que vous avez fait pour que ce défi fonctionne dans le fichier HTML et surtout dans le fichier JavaScript. Publiez-le et envoyez le lien URL de partage de ce screencast sur le forumulaire <https://forms.gle/D1fMdqtdam53JCiXA>