BAGGIO	Javascript	BTS
TP	Introduction	SNIR

Consignes

Objectifs

L'objectif est de réaliser, à l'instar d'un Cookie Clicker, un « Chips Smasher » : chaque clic permettant d'écraser des chips. Nous allons utiliser uniquement les technologies client pour cela : HTML, Javascript et CSS.

Le joueur peut suivre plusieurs éléments : le nombre de clics qu'il a réalisé, le nombre de chips écrasés et les bonus qu'il possède.

Chaque bonus est défini par un coût (en chips), un icône (en utilisant les emotes disponibles dans la section « Unicode blocks » à la page https://en.wikipedia.org/wiki/Emoji#Unicode_blocks) ainsi qu'un multiplicateur.

Le processus d'achat d'un bonus est donné en détail, avec un exemple des étapes, dans l'algorigramme présenté en Annexe X6.

Notation

Seront pris en compte dans la notation l'originalité du style, la réactivité des composants, l'algorithmie, le style général du code. Des critères non-exhaustifs sont donnés en exemple en Annexe.

Squelette de base

Créez trois fichiers : un fichier HTML (nommé index), un fichier CSS et un fichier Javascript. Pour le moment, ces fichiers seront vides.

Une fois ces trois fichiers créés, remplissez le fichier index avec le squelette de base d'un fichier HTML.

Puis, liez les fichiers styles et scripts à votre page.

Votre page principale HTML doit être découpée en trois parties :

- Un en-tête, contenant le nombre de clics effectués, le nombre de chips détenus, et les bonus actifs
- Un contenu principal, contenant les boutons d'actions (donc achats de bonus, et bouton principal de « smash »)
- Un pied de page, contenant le nom de votre équipe et la liste de ses membres

Ces trois parties utiliseront les éléments « header », « main » et « footer » de manière adéquate.

Style

Dans votre page de style, remettez à zéro les marges extérieures et intérieures, et placez une couleur de fond. Modifiez votre style pour que l'en-tête de page prenne toute la largeur, de même que le pied de page. Ce dernier devra également se « coller » en bas de la page, quelque soit la hauteur de la fenêtre.

Le contenu principal devra être centré en largeur.

Le design global est laissé à votre discrétion, mais doit être utilisable et doit montrer une certaine originalité.

Contenu

Formulaire principal

Dans la partie principale, vous allez devoir rajouter au moins un bouton : le clic principal, ou « Smash ». Chaque clic sur ce bouton va incrémenter le nombre de clics, et augmenter le nombre de chips écrasés d'un certain nombre, dépendant des bonus actifs.

Dans cette même partie principale, vous ajouterez également des boutons pour acheter les bonus.

Les boutons ne devant pas provoquer de changement de page, il est conseillé d'utiliser des balises « button » et de ne pas utiliser de balise « form ».

Il est possible de placer une image en fond de bouton, en utilisant la directive CSS background-image, mais vous pouvez aussi utiliser un input type image : https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/image .

Contenu de l'en-tête

L'en-tête de page doit afficher :

- Le nombre de clics effectués
- Le nombre de chips écrasés
- Les bonus actifs

Le plus simple pour cela est, pour chaque donnée à ajouter, d'utiliser une balise de type span (ou autre) mais surtout de leur indiquer à chacune un attribut « id » (qui doit donc être unique). Un exemple est donné ci-contre.

```
<p>
    Nombre de clics:
    <span id="clicks">0</span>
</p>
```

Dynamisme et fonctionnement

Vous allez devoir utiliser au moins quatre variables globales :

- Une variable pour retenir le nombre de clics
- Une variable pour retenir le nombre de chips écrasés
- Une variable pour retenir les bonus achetés
- Une variable pour lister les bonus possibles

Dans une première partie, concentrez-vous uniquement sur la base : le clic et la comptabilisation du clic.

Pour cela, commencez par attacher un événement au chargement de la page, afin d'exécuter le code d'initialisation. Pour rappel, le plus simple est d'utiliser des fonctions pour les lier à un événement capturé.

Dans ce code d'initialisation, attachez un événement au clic sur le bouton principal.

N'oubliez pas : lorsqu'une variable est définie hors d'une fonction en utilisant « var », elle est disponible partout : c'est une variable dite « globale ».

Incrémenter les variables est une chose : il faudra aussi mettre à jour l'affichage, donc le contenu des balises dans l'en-tête.

Annexes

Annexe X1 : Inclusion de styles

Il existe trois manières d'inclure du CSS : en ligne, en tant que définition de styles ou via un fichier externe.

X1.1 : En ligne

Ce type d'inclusion est à éviter de manière générale, et n'est utile en général que lorsque le style est généré dynamiquement, comme avec PHP. Il s'agit de placer, sur l'élément HTML que vous voulez styler, un attribut « style » et d'y indiquer les propriétés et valeurs CSS voulues.

```html
<p style="font-style: italic; color: #fad700;">
    Réalisé par M. Chassel
</p>
```

1 - Ce paragraphe sera en italique, de couleur dorée

X1.2 : Définition de styles

Ce type d'inclusion est souvent à éviter, et n'est utile que pour surcharger des styles pour une page particulière, ainsi que lorsque du style est généré dynamiquement, comme avec PHP.

```html
<head>
    <title>Test clicker</title>
    <meta charset="utf-8" />
    <style type="text/css">
        html, body {
            margin: auto;
            padding: 0px;
        }

        p, abbr {
            font-style: italic;
            color: #fad700;
        }
    </style>
</head>
```

Il s'agit de placer une balise « style » dans l'en-tête du document HTML, et d'y insérer les directives CSS (donc sélecteurs, propriétés et valeurs CSS).

Il est préférable, lors de l'utilisation de la balise « style », d'indiquer l'attribut « type ». Celui-ci indique le type MIME du contenu associé : ici « text/css ».

X1.3 : Fichier externe

Ce type d'inclusion est le plus couremment utilisé, et celui à privilégier.

Il s'agit de placer une balise « link » dans l'en-tête du document HTML, d'indiquer l'attribut « rel » à la valeur « stylesheet » et l'attribut « href » en indiquant le chemin vers la feuille de style (donc le fichier CSS lié).

```html
<link rel="stylesheet" href="./style.css" />
```

2 - Exemple classique d'inclusion d'une feuille de style externe

Il est possible, en utilisant l'attribut « media », de préciser différentes feuilles de styles en fonction de l'outil utilisé pour lire le document HTML : navigateur internet (valeur « screen »), impression (valeur « print »), et même d'y indiquer des conditions complexes, comme la taille de l'écran.

La balise « link » est complexe, et permet beaucoup de choses. Davantage de contenu est disponible sur la documentation MDN sur le sujet : https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link

Annexe X2 : exécuter du code Javascript

Le code javascript peut être exécuté de nombreuses manières en fonction de ce qui est désiré. Chaque événement HTML peut être « capturé » pour exécuter du code Javascript.

X2.1 : Exemple de capture d'événement

La balise « button » permet d'afficher un bouton générique. Lorsque l'on clique sur ce bouton, un événement « click » se déclenche et se répand, depuis le bouton vers toute la page.

Il est possible de « capturer » cet événement sur une balise HTML, en indiquant un attribut « onclick », et en définissant sa valeur au code Javascript à exécuter, comme ci-dessous.

```
<button onclick="console.log('Evénement clic sur le bouton !');">Tester !</button>
```

Dans cet exemple, lorsque le bouton sera cliqué, un message s'affichera dans la console.

La majorité des événements HTML/Javascript fonctionne de la même manière. Par exemple, si nous modifions notre code pour l'événement « mouseover » qui se déclenche lorsque nous passons la souris sur un élément :

```
<button onmouseover="console.log('Evénement mouveover sur le bouton !');">Tester !</button>
```

Nous avons donc placé un attribut « onmouseover » pour capturer l'événement « mouseover ».

X2.2 : Exécution de code au lancement de la page

Le code Javascript peut aussi être placé via la balise « script » avec l'attribut « type » indiquant son type MIME : « text/javascript ». Cette balise peut être placée dans le corps de la page ou dans l'en-tête et s'exécutera dès qu'il est chargé.

```
<!doctype html>
<html>
    <head>
        <title>Test clicker</title>
        <meta charset="utf-8" />
        <script type="text/javascript">
            console.log('chargement en-tête');
        </script>
    </head>
    <body>
        <script type="text/javascript">
            console.log('chargement début de page');
        </script>
        <header>…
        </header>
        <main>…
        </main>
        <footer>…
        </footer>
        <script type="text/javascript">
            console.log('chargement fin de page');
        </script>
    </body>
</html>
```

Au lieu de placer le contenu du code Javascript entre les balises, vous pouvez (et c'est la façon recommandée) indiquer un fichier Javascript contenant votre code en utilisant l'attribut « src » comme dans l'exemple ci-dessous.

```
<script type="text/javascript" src="./script.js"></script>
```

Dans l'exemple ci-contre (le contenu de la page a été masqué), nous avons inclus trois fois du Javascript. Ce code écrit dans la console lorsqu'il est chargé et exécuté.

```
chargement en-tête
chargement début de page
chargement fin de page
```

Mais cet ordre d'exécution peut varier selon les cas. Il est donc déconseillé de se fier à l'ordre d'inclusion du code Javascript dans le HTML ; il est préférable d'utiliser l'événement « load » sur la page HTML : celui-ci est lancé lorsque la page HTML est entièrement chargée.

```
<body onload="console.log('page entièrement chargée !');">
```

```
chargement en-tête
chargement début de page
chargement fin de page
page entièrement chargée !
```

X2.3 : Bubbling

Quand vous cliquez sur un élément de la page, l'événement de clic ne se déclenche pas que sur cet élément : il se propage.

Dans l'exemple ci-contre, nous avons placé du code Javascript sur le click au button, sur la balise main, et sur le body.

```
    <meta charset="utf-8" />
</head>
<body  onclick="console.log('body');">
    <header>···
    </header>
    <main onclick="console.log('main');">
        <button onclick="console.log('button');">Tester !</button>
    </main>
    <footer>···
    </footer>
</body>
```

Lors du clic sur le bouton, l'événement de clic se propage : d'abord depuis l'élément sur lequel nous avons directement cliqué, puis à travers ses parents (donc main, et body).

```
button
main
body
```

Ce phénomène de propagation est appellé « bubbling », ou « bouillonnement ».

Il est possible de stopper cette propagation si nécessaire, comme dans l'exemple ci-dessous.

```
<button onclick="console.log('button'); event.stopPropagation();">Tester !</button>
```

X2.4 : Attacher des événements

Ajouter un attribut sur chaque élément pour capturer des événements est une solution « rapide », mais elle est déconseillée pour plusieurs raisons :

- Le code (HTML et Javascript) devient difficile à lire, est mélangé et donc confus
- Le code est limité à UN élément sur la page
- On est très limité en code qu'on peut placer (peu d'espace, sauts de ligne…)

HTML / Javascript possède une méthode de base pour « attacher » du code Javascript à un événement : addEventListener.

Dans l'exemple ci-contre, nous avons « attaché » l'appel à une fonction « iAmLoaded » à l'événement « load » de la fenêtre. Ainsi, au chargement de la page, la fonction « iAmLoaded » est appellée.

```
    <script type="text/javascript">
        function iAmLoaded() {
            console.log('Yes, i am loaded !');
        }
        window.addEventListener('load', iAmLoaded);
    </script>
</head>
```

X2.5 : Accéder à un élément HTML pour y attacher des événements

La fonction getElementById est une méthode Javascript pour accéder à un élément HTML et interagir avec, en utilisant son attribut « id ».

L'exemple ci-contre attache une fonction au clic sur l'élément ayant l'ID « testButton ».

```
<main>
    <button id="testButton">Tester !</button>
</main>
<footer>···
</footer>
<script type="text/javascript">
    function doIt() {
        console.log('Bouton cliqué !');
    }
    document.getElementById('testButton').addEventListener('click', doIt);
</script>
```

Notez que l'événement a été attaché en fin de page : en effet, nous avons besoin que l'élément existe (donc que la page ai fini de charger cet élément) pour pouvoir lui attacher un événement.

Il est donc généralement préférable d'exécuter le code Javascript au plus tard, une fois que toute la page HTML ai fini de charger.

X2.6 : Modifier le contenu d'une balise

Il est possible en Javascript de modifier le contenu d'une balise (donc ce qu'elle contient).

On commence par sélectionner l'élément, en utilisant par exemple la méthode getElementById.

Ensuite, deux propriétés « réactives » (dites « setters ») existent :

- innerHTML permet de modifier le contenu HTML, donc d'inclure du code HTML dynamique
- innerText permet de modifier le contenu texte, donc d'inclure uniquement du texte

Observez l'exemple ci-contre :

```
document.getElementById('bonus').innerText = 'Aucun bonus';
```

Nous sélectionnons l'élément ayant l'id « bonus ». Puis nous remplaçons son contenu par le texte « Aucun bonus ». Ainsi, tout ce qui est entre les deux balises span de l'exemple ci-contre

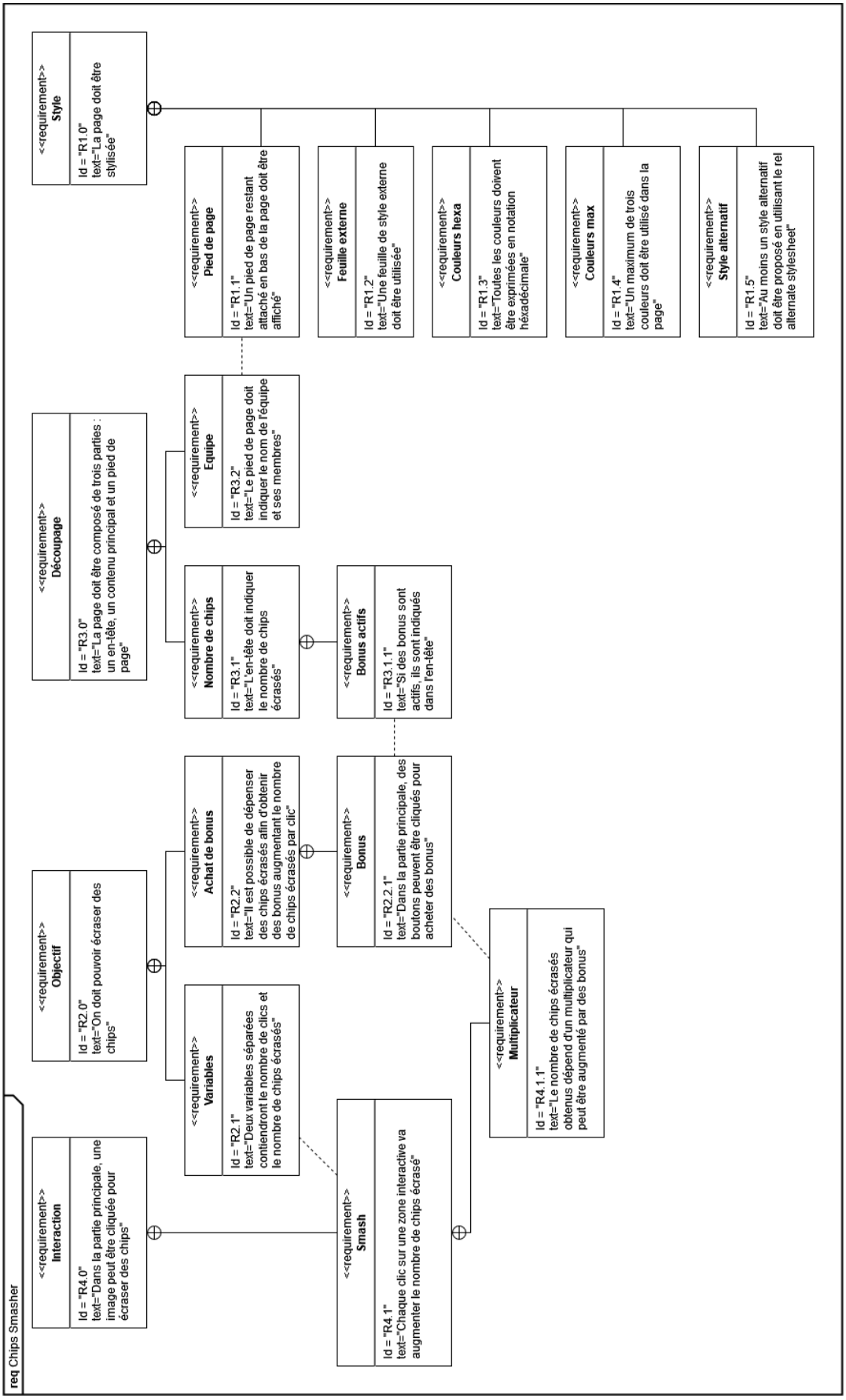
```
<span id="bonus"></span>
```

seront remplacés par le texte « Aucun bonus ».

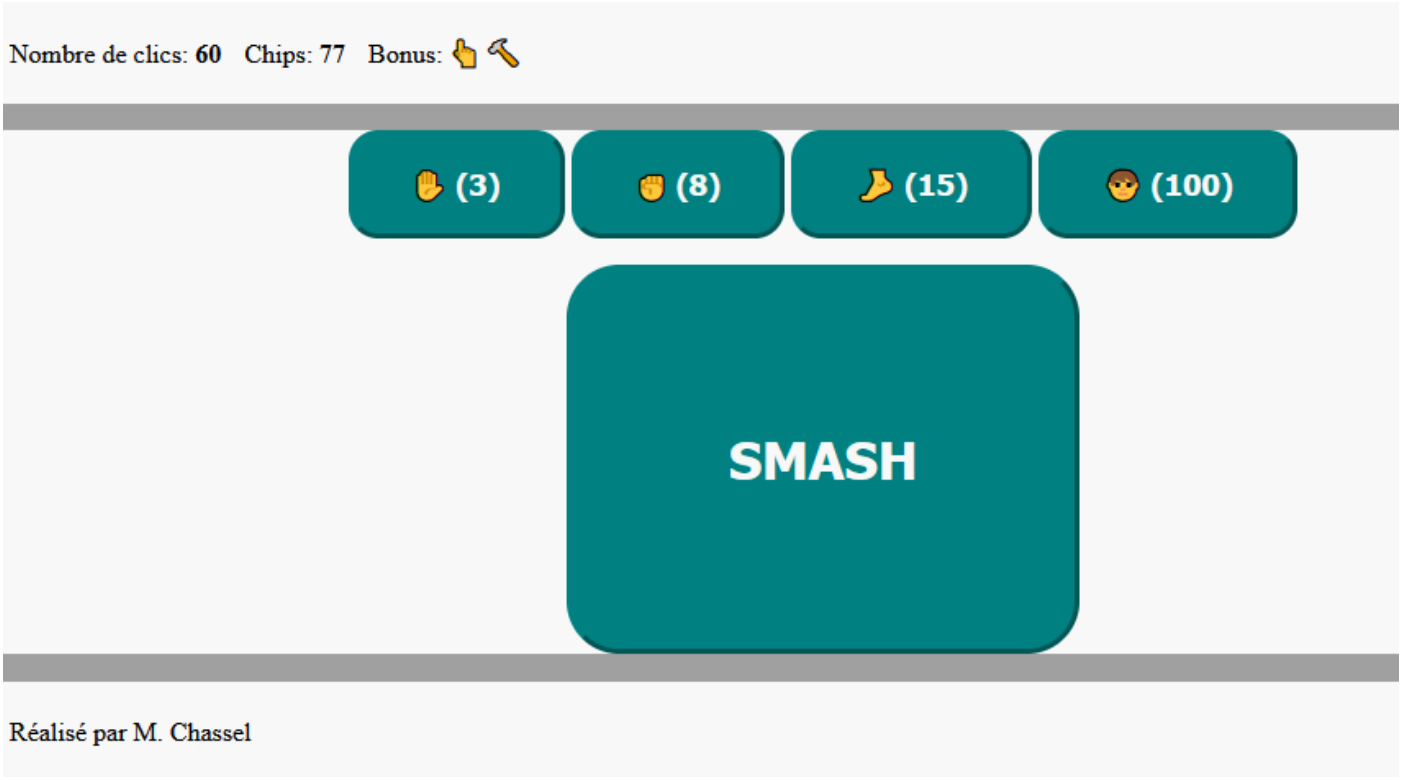
Ci-dessous, un autre exemple montre la même utilisation, de innerHTML cette fois, pour remplacer le contenu par une balise strong de couleur rouge :

```
document.getElementById('bonus').innerHTML = '<strong style="color: red;">Aucun bonus</strong>';
```

Annexe X3 : Diagramme d'exigences



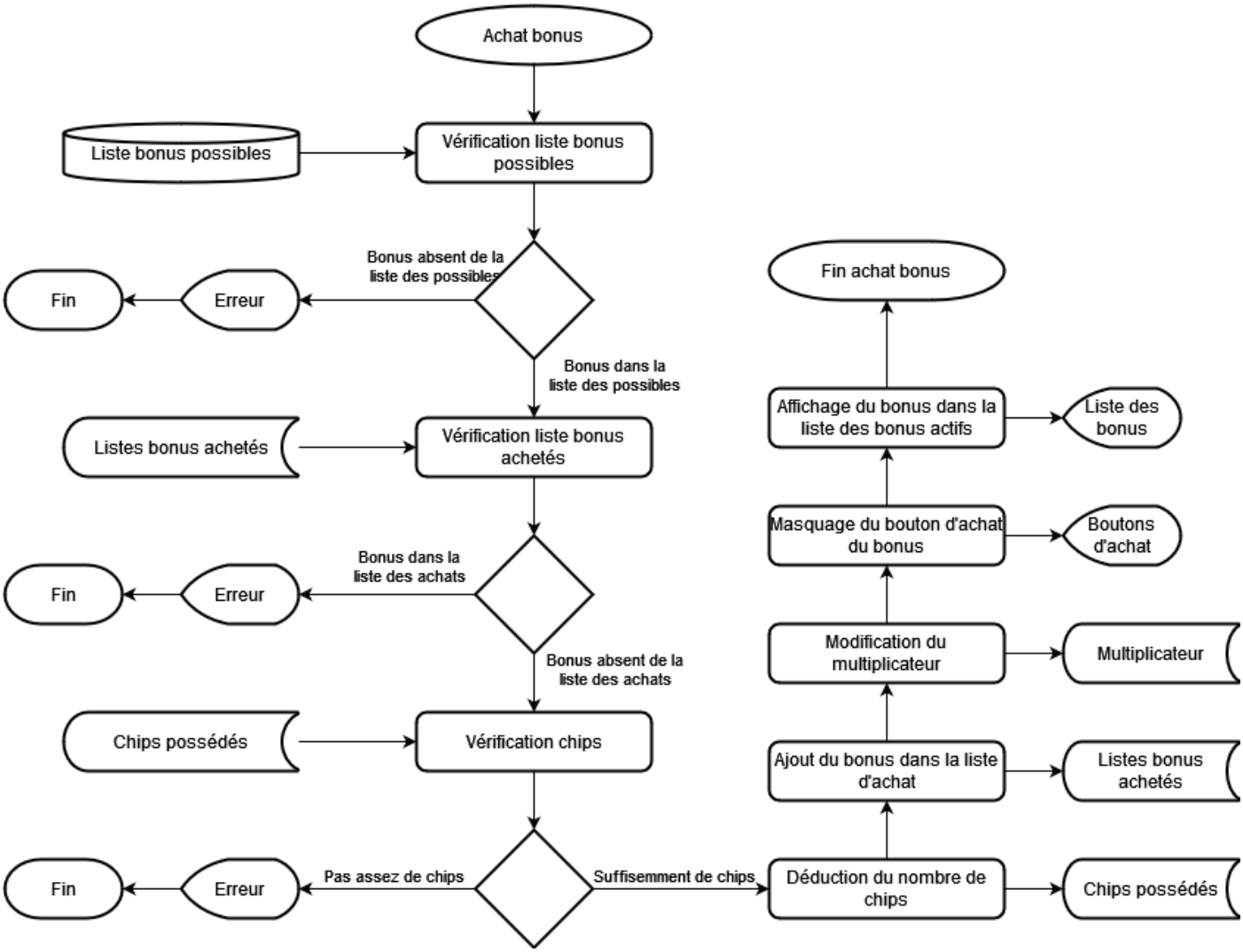
Annexe X4 : exemple (moche et non-exhaustif) de rendu



Annexe X5 : exemple de notation (purement indicatif)

Catégorie	Point	Barême
Algorithmie	Code clair : pas de code mort, relecture aisée	10
	Variables : nommage cohérent, utile, portée (var/let)	5
	Méthodes : utilisation et création	5
	Evénements : utilisation adéquate	5
	Commentaires intelligents bien placés	5
Style	Originalité	5
	Couleurs : maximum de trois, température	5
	Images : utilisation adéquate, appropriée	5
	Design : lisibilité, utilisabilité	5
HTML	Structure : structure valide et correcte	10
	Balises : HTML5, écriture correcte	5
	Gestion dynamique : insertion par Javascript	5
Global	Respect des consignes	15
	Respect du diagramme d'exigences	15

Annexe X6 : algorigramme exemple pour l'achat de bonus



Annexe X7 : code exemple pour lister les bonus possibles

```
var bonusDb = {
    'finger': {
        'icon': '👆',
        'cost': 1,
        'multiplier': 0.1
    },
    'hand': {
        'icon': '🖐',
        'cost': 3,
        'multiplier': 0.2
    },
    'fist': {
        'icon': '✊',
        'cost': 8,
        'multiplier': 0.3
    },
    'foot': {
        'icon': '🦶',
        'cost': 15,
        'multiplier': 0.5
    },
    'hammer': {
        'icon': '🔨',
        'cost': 50,
        'multiplier': 1
    },
    'friend': {
        'icon': '👨',
        'cost': 100,
        'multiplier': 1
    },
};
```

Annexe X8 : Proposition de diagramme de Gantt