
Développement d'outils ou matériel d'enseignement de l'informatique

Visualisation interactive des opérations et algorithmes sur les listes à l'aide d'un jeu de cartes

Collège du sud, Travail de maturité Version finale

Grégoire Geinoz

avr. 03, 2022

Table des matières

1	Introduction	1
1.1	Motivation personnelle	1
1.2	Présentation de l’outil	1
1.3	Intérêt de l’outil	2
1.4	Technologies utilisées	2
1.5	Configuration matérielle requise pour utiliser l’outil	2
1.6	Connaissances requises pour utiliser l’outil	2
1.7	Connaissances requises au développement de l’outil	3
1.8	Difficultés anticipées	3
2	API	5
2.1	Fonctionnement	5
3	Scénarios d’utilisation	7
3.1	Introduction aux listes	7
3.2	Introduction aux algorithmes	7
4	Aspects pédagogiques	11
4.1	Constructivisme	11
4.2	Modèle d’apprentissage de l’outil	11
4.3	En quoi l’outil se distingue	12
4.4	Potentiel du projet	12
5	Fonctionnement de l’outil	13
5.1	L’utilisation de Phaser	13
5.2	Système d’animation	17
6	Conclusion	25
6.1	Evolution du projet	25
6.2	Difficultés rencontrées lors du développement	26
6.3	Scénarios pédagogiques	26
6.4	Pistes d’amélioration	26
7	Tester l’outil	29
7.1	Télécharger le dépôt GitHub	29
7.2	Ouvrir un serveur HTTP local	29
7.3	Fonctions disponibles de l’API	30

7.4	Cartes	30
7.5	Variables	32
8	Glossaire	35
9	Annexe	37
9.1	Code source	37
10	Déclaration personnelle	55
	Index	57

1.1 Motivation personnelle

Le choix du sujet de ce travail résulte de mon intérêt inlassable de toujours en apprendre plus sur les sujets qui me passionnent. Parmi ceux-ci, l’informatique ou plus précisément la programmation m’a toujours fasciné. Ainsi, si mon travail permet d’aider à appréhender et à comprendre des concepts qui semblent abstraits ou difficiles au premier abord, c’est naturellement que je m’impliquerai dans le projet. C’est donc avec le souhait de rendre un peu plus accessible le monde de la programmation que j’ai fait le choix de mon sujet.

1.2 Présentation de l’outil

L’objet de ce travail de maturité consiste en la programmation d’un outil permettant d’accompagner un professeur souhaitant développer une compréhension intuitive de la notion de liste à ses élèves. Il permet d’avoir une vision claire de ce qu’est une liste en programmation et quelles sont les manipulations qu’un ordinateur peut effectuer sur celle-ci.

Concrètement, une liste est représentée comme étant un “jeu de cartes”. Chaque élément de la liste est représenté une carte avec sa valeur affichée d’un côté de la carte. Ainsi, tous les éléments de la liste forment le jeu de cartes au complet. Certains algorithmes requièrent des variables externes à la liste. Ces variables sont créées via des appels de fonctions sur une API¹ et sont représentées dans une zone de l’écran spécifique. Cependant, des variables “spéciales”, qui n’ont comme seul rôle d’être un indice qui parcourt les éléments d’une liste (dans une “boucle for” par exemple), auraient également dû pouvoir être créées et auraient dû être représentées comme des flèches qui pointent sur l’élément de la liste qui correspond à l’indice contenu dans la variable.

Le but étant de démontrer les actions opérées par un ordinateur sur une liste, les interactions entre les éléments de la liste entre eux ou avec une variable sont animées. Ainsi, pendant l’exécution d’un programme, ce dernier fera discrètement appel à l’API, afin qu’une animation se déclenche et montre sur l’écran le comportement de la liste.

Malheureusement, toutes les fonctionnalités prévues initialement n’ont pas pu être développées avant la fin du travail. De plus, certaines fonctionnalités ont simplement été soustraites au cahier des charges, par exemple : la possibilité de déplacer les cartes avec la souris, de garder en mémoire un historique de toutes les actions effectuées afin de rejouer la série d’instructions, ou même la possibilité d’écrire du code directement dans un éditeur juxtaposé à la zone de rendu

1. Voir chapitre *API*

des cartes. Bien qu'intéressantes, ces fonctionnalités ne sont pas essentielles au projet et ne sont donc envisageables que dans un second temps.

1.3 Intérêt de l'outil

Les listes et les algorithmes sont des sujets importants à comprendre et à maîtriser dans le cadre de l'apprentissage de la programmation. Actuellement, ces concepts sont amenés aux étudiants et expliqués de manière bien trop abstraite. Il est nécessaire de comprendre ce que représente une liste pour un ordinateur et comment il peut interagir avec ; ainsi que de comprendre le fonctionnement des algorithmes, étape par étape.

Des plateformes d'enseignement de l'informatique tel que "code.org" ont un concept similaire au projet de ce travail de maturité : développer une compréhension intuitive de concepts par l'expérimentation. Cependant, leur projet cible un public jeune pour leur inculquer des bases de programmations par exemple en posant brique par brique des blocs de code afin d'amener un personnage jusqu'à son objectif. Un autre programme est intéressant car il permet d'expérimenter des algorithmes de tri en effectuant des comparaisons manuellement entre des boîtes et en essayant de les classer des plus légères aux plus lourdes avec le moins de comparaisons possible. Le programme est disponible sur le site « <https://www.csfieldguide.org.nz/en/interactives/sorting-algorithms/> ». Enfin, il y a également ce site « <https://professeurb.github.io/articles/tris/> », qui est parsemé d'animations qui démontrent le fonctionnement de plusieurs algorithmes de tri. Cependant, le projet envisagé dans le cadre de ce travail de maturité se distingue des projets similaires par sa capacité à s'adapter à de multiples scénarios et grâce à son système d'API qui permet un parallèle entre les animations et l'exécution d'un code concret.

1.4 Technologies utilisées

Afin que l'utilisation de l'outil soit la plus accessible possible, les langages de programmation du Web ont été utilisés pour le développer. Le projet repose donc sur 2 technologies principales : Javascript et HTML5. Cependant pour alléger la quantité de travail, un framework² de jeu 2D, Phaser³, est utilisé pour gérer tout ce qui relève de l'affichage d'images, de la gestion de scènes et d'événements et autres qui impliquent la gestion d'objets (les cartes et les variables). Enfin, le projet étant écrit en Javascript principalement, un interpréteur de code peut être nécessaire si le langage étudié par l'outil est différent.

1.5 Configuration matérielle requise pour utiliser l'outil

Comme l'outil a été développé dans un souci de portabilité, il repose sur les technologies du Web. Ainsi, seul un accès à un ordinateur opérationnel étant doté d'un navigateur internet et d'une connexion est nécessaire pour utiliser l'outil.

1.6 Connaissances requises pour utiliser l'outil

L'outil fait appel aux notions de variables ; éléments qui constituent la liste, ainsi qu'aux opérations de base entre variables numériques (calculs, affectations de valeurs).

2. Voir glossaire

3. <https://phaser.io/>

1.7 Connaissances requises au développement de l'outil

Afin de comprendre le fonctionnement de l'outil et pouvoir continuer son développement, un niveau de base qui couvre tous les fondamentaux du Javascript est nécessaire. Le code étant largement commenté, il n'est pas forcément essentiel de savoir utiliser le framework Phaser pour comprendre le code source.

1.8 Difficultés anticipées

Aux prémices du développement, les potentielles difficultés anticipées ont été de définir les fonctionnalités principales du projet afin d'avoir un outil capable de réagir à une grande variété de scénarios et de les implémenter de manière à en faire une base solide sur laquelle le projet peut reposer sans devoir constamment retravailler la base. Il s'agissait ici de prévoir des scénarios d'utilisation concrets afin d'orienter le projet et de définir quelles étaient les fonctionnalités essentielles.

L'API proposée par l'outil est l'essence même qui lui donne son sens. En effet, c'est elle qui permet l'interaction de l'utilisateur avec le programme principal. De plus, une API peut être implémentée dans n'importe quel programme, ce qui permet à l'outil de s'adapter, au niveau des interactions, aux convenances de celui qui veut l'utiliser.

2.1 Fonctionnement

Une API, abréviation de « Application Programming Interface », ou en français « interface de programmation d'application », désigne une façade exposant des fonctions qui peuvent être appelées depuis un autre code. Elle permet d'intégrer le programme dans une application quelconque :

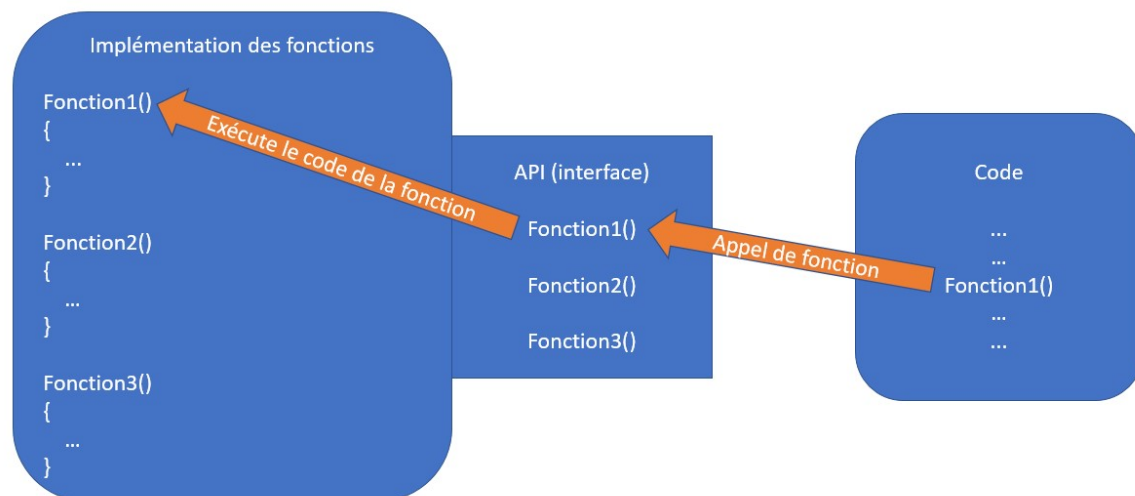


Fig. 1 – Fonctionnement de l'API.

CHAPITRE 3

Scénarios d'utilisation

Ce chapitre vise à apporter des idées concrètes quant à l'utilisation du projet par un professeur pour ses élèves.

3.1 Introduction aux listes

Un premier scénario, et pas des moindres, est de faire expérimenter aux élèves toutes les opérations qu'il est possible de réaliser avec des listes : ajouter des éléments, les supprimer, les déplacer, lire leur valeur, les comparer, etc.

Les expérimentations se font directement en écrivant du code dans un langage quelconque, tant qu'il permet d'appeler les fonctions de l'API au moment d'une opération.

3.2 Introduction aux algorithmes

Si les notions de bases de la gestion des listes sont acquises, alors l'outil devrait être en tout cas partiellement adapté à démontrer visuellement les opérations effectuées par un algorithme reposant sur l'utilisation d'une liste :

3.2.1 Sommer tous les éléments d'une liste

```
1 // Liste quelconque.  
2 const liste = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
3  
4 // La somme est nulle au départ.  
5 let somme = 0;  
6  
7 // Parcourt la liste.  
8 for (let i = 0; i < liste.length; i++)  
9 {  
10     somme += liste[i];
```

(suite sur la page suivante)

(suite de la page précédente)

```
11 }  
12  
13 // Affiche la somme dans la console.  
14 console.log("La somme est : " + somme);
```

Dans cet algorithme certaines lignes peuvent faire appel à l'API afin d'animer l'exécution du programme et de mieux comprendre les opérations effectuées :

- Ligne 2 : `cards.appendList(liste)` anime la création de la liste initiale.
- Ligne 5 : `vars.create("somme", 0)` anime la création de la variable `somme` avec sa valeur de 0.
- Ligne 10 : `vars.add("somme", i)` ajoute la valeur de la carte à la position `i` à la valeur de la variable `somme`.

3.2.2 Trouver le minimum

Cet algorithme simple à mettre en place, permet d'avoir un avant-goût du potentiel de l'outil. Sur une liste non triée, voici à quoi ressemblerait un algorithme qui recherche la plus petite valeur dans une liste :

```
1 // List non-triée.  
2 const liste = [3, 7, 2, 9, 1, 8, 4, 6, 5];  
3  
4 // Le minimum est un nombre de la liste.  
5 let minimum = liste[0];  
6  
7 // Parcourt la liste.  
8 for (let i = 0; i < liste.length; i++)  
9 {  
10     if (liste[i] < minimum)  
11     {  
12         minimum = liste[i];  
13     }  
14 }  
15  
16 // Affiche le minimum dans la console.  
17 console.log("Le minimum est : " + minimum);
```

- Ligne 2 : `cards.appendList(liste)` anime la création de la liste initiale.
- Ligne 5 : `vars.create("minimum")` et `cards.read(0, "minimum")` anime la création de la variable `minimum` et lui assigne la valeur de la carte à l'indice 0.
- Ligne 10 : Il aurait été intéressant de pouvoir montrer visuellement la condition mais cette fonctionnalité ne figure pas dans la version actuelle de l'outil.
- Ligne 12 : `cards.read(i, "minimum")` assigne la valeur de la carte à l'indice `i` à la variable `minimum`.

3.2.3 Recherche dichotomique

La recherche dichotomique fait partie des algorithmes intéressants auxquels l'outil n'apporte pas encore de réels soutiens visuels qui pourraient aider à la compréhension, les fonctionnalités cruciales qu'il manque à l'outil sont la possibilité de créer des variables spéciales dont la valeur est un indice qui désigne une carte de la liste, et la visualisation des conditions :

```
1 // La liste passée en paramètre doit être triée.
2 function recherche(liste, valeur)
3 {
4     // Borne inférieure prise en compte dans la recherche.
5     let premier = 0;
6
7     // Borne supérieure prise en compte dans la recherche.
8     let dernier = liste.length - 1;
9
10    // Déclarations de variables utiles.
11    let milieu;
12    let element;
13
14    while (premier <= dernier)
15    {
16        // Milieu des bornes inférieure et supérieure.
17        milieu = Math.floor((premier + dernier) / 2);
18
19        // Element courant de la liste.
20        element = liste[milieu];
21
22        if (element == valeur)
23        {
24            // Retourne l'indice de l'élément trouvé.
25            return milieu;
26        }
27        else if (element < valeur)
28        {
29            // Décale la borne inférieure après le milieu.
30            premier = milieu + 1;
31        }
32        else
33        {
34            // Décale la borne supérieure avant le milieu.
35            dernier = milieu - 1;
36        }
37    }
38
39    // Retourne -1 si la valeur n'a pas été trouvée.
40    return -1;
41 }
42
43 // Affiche les résultats de recherches.
44 console.log(recherche([1, 2, 3, 4, 5, 6, 7, 8, 9], 5)); // 4
45 console.log(recherche([1, 2, 3, 4, 5, 6, 7, 8, 9], 10)); // -1
```

Aux lignes 5, 8 et 11, le mieux aurait été de pouvoir créer des variables spéciales de type « indice » qui montre

visuellement à quelles cartes elles font référence dans la liste. De plus, aux lignes 14, 22 et 27, l'animation des conditions ne figurent pas encore comme fonctionnalités de l'outil :

- Ligne 5 : `vars.create("premier", 0)` anime la création de la variable `premier`.
- Ligne 8 : `vars.create("dernier", liste.length - 1)` anime la création de la variable `dernier`.
- Ligne 11 : `vars.create("milieu")` anime la création de la variable `milieu` avec une valeur par défaut.
- Ligne 12 : `vars.create("element")` anime la création de la variable `element` avec une valeur par défaut.
- Ligne 17 : `vars.assign("milieu", Math.floor((premier + dernier) / 2))` anime l'assignation de la valeur à la variable `milieu`.
- Ligne 20 : `vars.assign("element", liste[milieu])` anime l'assignation de la valeur à la variable `element`.
- Ligne 30 : `vars.assign("premier", milieu + 1)` anime l'assignation de la valeur à la variable `premier`.
- Ligne 35 : `vars.assign("dernier", milieu - 1)` anime l'assignation de la valeur à la variable `dernier`.

CHAPITRE 4

Aspects pédagogiques

Ce chapitre traite un aspect des sciences de l'éducation qui tend à montrer que l'on assimile mieux les concepts en les découvrant par soi-même, et en expérimentant.

4.1 Constructivisme¹

Le constructivisme est un modèle d'apprentissage développé par *Jean Piaget*. Il pense que toute nouvelle connaissance est basée sur des connaissances déjà acquises ; que les connaissances sont construites et adaptées à partir de ce que l'on sait déjà, d'où son nom « constructivisme ».

Selon *Piaget*, la connaissance naît des interactions d'un sujet avec un objet. Ainsi, l'environnement doit amener le sujet à se confronter à un scénario qu'il ne connaît pas et qui demande des connaissances nouvelles afin d'être compris. De cette manière, le sujet peut interagir avec l'objet du scénario et découvrir un concept par lui-même en le basant sur ce qu'il sait déjà pour finalement se créer ou affiner une représentation personnelle du concept.

4.2 Modèle d'apprentissage de l'outil

L'outil développé dans le cadre de ce travail base son modèle d'apprentissage sur la philosophie constructivisme. En effet, tous les concepts qu'il permet d'aborder nécessitent l'utilisation d'une liste, qui est représentée tel un objet réel, sous la forme d'un « jeu de cartes ». Avec cet objet de base, selon la façon dont l'API est intégrée, un étudiant peut interagir avec la liste en appelant indirectement des fonctions de l'API à travers du code, ou constater les effets qu'ont certaines lignes de code sur la liste. Ainsi, l'étudiant peut découvrir des concepts par lui-même sur la base de ce qu'il connaît déjà (les notions de variables et les opérations entre variables numériques).

1. Selon Piaget « https://www.youtube.com/watch?v=mcjY4KU1HWM&ab_channel=CentreJeanPiaget »

4.3 En quoi l'outil se distingue

Comme cité dans l'introduction, il existe des outils similaires à celui développé ici. Cependant, celui-ci se distingue des autres en grande partie grâce à sa portabilité, à sa personnalisation et à son large spectre d'applications. En effet, il peut être implémenter par n'importe qui dans un langage quelconque tant qu'il est possible d'appeler des fonctions de l'API discrètement pendant l'exécution du code ou après, instruction par instruction. L'API pourrait même être implémentée dans un projet plus original basé sur des suites d'instructions représentées par des briques, comme dans un des projets de "code.org"². Enfin, il peut faire face à divers scénarios afin de visualiser des algorithmes ou tout simplement des instructions sur une liste afin de comprendre comment elle réagit. En somme, ce qui distingue l'outil de ce qui existe déjà dans le même registre, dans le monde de la pédagogie, est qu'il permet d'expliquer des concepts variés et qu'il peut être implémenté selon la manière d'enseigner qui convient le mieux aux besoins d'un professeur.

4.4 Potentiel du projet

Le projet possède certainement un potentiel encore faiblement exploité qui le dépasse. En effet, il n'est encore qu'à un stade peu évolué au niveau des animations et de son spectre d'applications qui peut être largement élargi. Le concept de l'outil pourrait être transposable pour visualiser une multitude de notions plus ou moins complexes, en gardant l'aspect de portabilité et de personnalisation au niveau de l'intégration de l'outil. Des concepts comme la récursivité ou la gestion de la mémoire en Python (qui possède un « garbage collector »³) sont des notions à priori complexes qui pourraient faire l'objet d'une extension intéressante de l'outil. Par exemple, les fonctions appelées récursivement pourraient être représentées comme une liste chaînée qui s'agrandit à chaque appel récursif et rétrécit à chaque fin de fonction.

2. Lien du projet : « <https://studio.code.org/s/aquatic/> »

3. *ramasse-miettes* en anglais

Fonctionnement de l'outil

Le projet repose sur plusieurs grands aspects, qui permettent une répartition du code en différents systèmes.

5.1 L'utilisation de Phaser

Phaser¹ se décrit comme un *framework*² de création de jeux vidéo, il contient donc de nombreux utilitaires facilitant la gestion d'images, les déplacements et des effets en tous genres comme la distorsion d'image ou des effets de transparence.

5.1.1 Système de scènes

L'un des plus gros avantages qu'offre Phaser est qu'il repose sur un système de scènes, qui possèdent des méthodes spécifiques permettant par exemple de précharger des images par la méthode *preload()* ou une méthode *update()* appelée plusieurs fois par seconde permettant d'actualiser des valeurs (très utile pour le système d'animation).

Afin de définir une scène, il faut créer une classe héritant de la scène de base de Phaser et d'implémenter ses méthodes :

```
class MainScene extends Phaser.Scene
{
    // Appelée lors de l'instanciation de la scène.
    constructor ()
    {
        super ("MainScene" );
    }

    // Méthodes native aux scènes de Phaser :

    // Principalement pour charger les images.
```

(suite sur la page suivante)

1. <https://phaser.io/>
2. Voir glossaire

(suite de la page précédente)

```
preload()
{
    // ...
}

// Principalement pour créer des objets de Phaser.
create()
{
    // ...
}

// Appelée à interval régulier
// pour actualiser des valeurs.
update()
{
    // ...
}
}
```

Pour instancier la scène, il suffit simplement de créer une variable de configuration afin de définir la taille du *canvas*³ utilisé pour dessiner les images, le nombre d'actualisation par seconde de la fonction *update()* ainsi que le nom de la classe de la scène principale, et d'instancier une classe *Phaser.Game* :

```
const config = {
    // taille du canvas.
    width: 1000,
    height: 600,

    // Nombre d'actualisation par seconde.
    fps: {
        target: 60,
        forceSetTimeout: true
    },

    // Scène principale.
    scene: [MainScene]
};

// Création du jeu avec les configurations.
const game = new Phaser.Game(config);
```

3. Voir glossaire

5.1.2 Système de gestion des cartes

Les scènes de Phaser permettent la création d'objets divers, comme des images, du texte, ou autre, à affichés à l'écran. Les cartes et les variables sont en fait des objets images dont les coordonnées varient afin de les faire se déplacer :

```
class MainScene extends Phaser.Scene
{
    preload()
    {
        // Charge l'image des cartes en mémoire.
        this.load.image("card", "images/card.png");
    }

    create()
    {
        // Crée une carte aux coordonnées (0; 0)
        // ayant comme image : "card".
        const card = this.add.image(0, 0, "card");
    }
}
```

5.1.3 Système d'événements

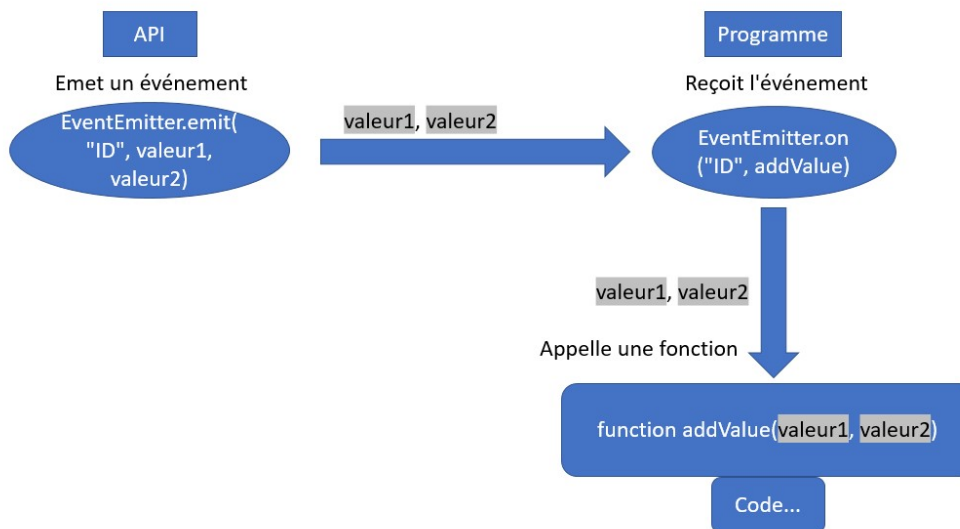


Fig. 1 – Fonctionnement du système d'événements de Phaser.

Le système d'événements de Phaser joue un rôle crucial dans le projet. En effet, il permet la communication entre l'API et le programme en lui-même. Pour ce faire, il est nécessaire d'instancier un objet *EventEmitter*⁴, proposé par Phaser. Cet objet permet d'émettre des événements et de les recevoir. C'est-à-dire qu'il est possible d'établir une connexion

4. <https://photonstorm.github.io/phaser3-docs/Phaser.Events.EventEmitter.html>

entre plusieurs fichiers ou parties de code différentes en émettant un événement contenant des paramètres qui seront transmis à une autre partie du code qui appellera une fonction en lui passant les paramètres spécifiés lors de l'envoi de l'événement. Cela revient, en résumé, à appeler une fonction qui est censée être hors de portée. Concrètement, le code se construit de la manière suivante :

Avant tout, il faut instancier l'objet *EventEmitter* :

```
// eventEmitter.js

const eventEmitter = new Phaser.Events.EventEmitter();
```

Ensuite, le code de l'API émet les événements :

```
// api.js

class Cards
{
    // Echange la position de deux cartes.
    swap(index1, index2)
    {
        // Emet l'événement.
        eventEmitter.emit(
            "swapCard", // Nom de l'événement.
            index1,     // Premier argument.
            index2      // Deuxième argument.
        );
    }
}
```

Enfin, le code du programme s'occupe d'intercepter l'événement :

```
// main.js

class MainScene extends Phaser.Scene
{
    create()
    {
        // Reçoit l'événement.
        eventEmitter.on(
            "swapCard", // Nom de l'événement.
            this.swapCard, // Fonction à appeler.
        );
    }

    // La fonction à appeler par l'événement.
    swapCard(index1, index2)
    {
        // ...
    }
}
```

5.2 Système d'animation

Le système d'animation permet au développeur de créer des schémas d'animation. C'est-à-dire que, par exemple, le développeur peut aisément créer une animation qui engendre le déplacement simultané ou séquentiel d'une ou plusieurs cartes.

5.2.1 Principe fondamental

Naïvement, on pourrait penser qu'il suffit que chaque carte possède une propriété *animation* qui contient les informations nécessaires à décrire une animation, par exemple, de déplacement :

```
const card = {...};

card.animation = {
  toAnimate: true,
  type: "movement",

  // Coordonnées où la carte doit se déplacer.
  x: ...,
  y: ...
};
```

Et qu'ainsi, dans la fonction *update()* gérée par Phaser, une *boucle for* parcourt toutes les cartes et effectue l'animation qui lui est attachée, si celle-ci doit être animée :

```
update() // Exécutée 60 fois par seconde par Phaser.
{
  // Parcourt toutes les cartes.
  for (let i = 0; i < nbCards; i++)
  {
    // Stocke la carte courante dans une variable.
    const currentCard = lstCards[i];

    if (currentCard.toAnimate)
    {
      // Gérer l'animation.
    }
  }
}
```

Or, cette manière de procéder comporte un gros désavantage. En effet, elle ne laisse au programme que la possibilité de gérer toutes les animations en même temps, ce qui signifie que si le développeur souhaite en jouer dans un certain ordre, il doit attendre que l'animation précédente soit terminée avant de configurer la suivante dans une ou plusieurs cartes. Ce n'est pas viable pour gérer une quantité importante d'animations qui s'exécutent à la suite.

C'est pour cela qu'un véritable système d'animation est nécessaire. Concrètement, la scène principale possède une propriété *animationQueue*. Il s'agit d'une liste initialement vide, qui stocke les animations les unes à la suite des autres. Ce principe simple permet de conserver l'ordre dans lequel les animations doivent être jouées ; selon l'ordre d'apparition dans la liste. Ce procédé nécessite donc également la création d'un objet *animation*, qui sera l'objet stocké dans la liste *animationQueue* :

```
function moveCard(targetCard, ...)
```

(suite sur la page suivante)

(suite de la page précédente)

```
const animation = {
  // Type de l'animation.
  type: "movement",

  // Contient une référence à la carte qui doit subir l'animation.
  card: targetCard,

  // Coordonnées où la carte doit se déplacer.
  x: ...,
  y: ...,

  // Devient vrai quand l'animation est arrivée à son terme.
  isFinished: false
};

// La liste d'animations relative à la scène principale.
animationQueue.push(animation);
}
```

De cette manière, la fonction `update()`, gérée par Phaser, peut accéder à la liste d'animations et les jouer dans l'ordre. Ainsi, une fois que la première animation de la liste est arrivée à son terme, il suffit de la supprimer pour pouvoir traiter la suivante.

```
update() // Exécutée 60 fois par seconde par Phaser.
{
  // Si la liste d'animations n'est pas vide.
  if (animationQueue.length > 0)
  {
    // Stocke la première animation de la liste.
    const currentAnimation = animationQueue[0];

    // Identifie la type de l'animation.
    switch (currentAnimation.type)
    {
      // Type: Déplacement
      case "movement":
        // Récupère la référence de la carte concernée.
        const card = currentAnimation.card;

        // Exécution de l'animation...
        // ...
        // ...

        if (/* Animation terminée */)
        {
          currentAnimation.isFinished = true;
        }
        break;
      // ...
    }

    // Si l'animation courante est terminée.
  }
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
    if (currentAnimation.isFinished)
    {
        // Retire la première animation de la liste.
        animationQueue.shift();
    }
}
```

Cependant, comme chaque objet *animation* décrit le comportement d'une unique carte, il n'est pas possible d'exécuter plusieurs animations simultanément. Pour contourner ce problème, le système est programmé pour jouer toutes les animations à la suite qui ne sont pas interrompues par un objet *animation* de type *break*⁵. Lorsque ce type est rencontré, le système s'assure que toutes les animations précédentes sont terminées avant de jouer le bloc d'animations suivant :

```
update() // Exécutée 60 fois par seconde par Phaser.
{
    // Parcours les animations de la liste (animationQueue).
    for (let i = 0; i < animationQueue.length; i++)
    {
        // Stocke l'animation courante de la liste.
        const currentAnimation = animationQueue[i];

        // Identifie le type de l'animation.
        if (currentAnimation.type == "movement")
        {
            // Récupère la référence de la carte concernée par l'animation.
            const card = currentAnimation.card;

            // Exécution de l'animation...
            // ...
            // ...

            if (/* Animation terminée */)
            {
                currentAnimation.isFinished = true;
            }
        }
        // ...
        else if (currentAnimation.type == "break")
        {
            // Si le type break est le premier élément de la liste.
            if (i == 0)
            {
                // Retire le premier élément de la liste.
                animationQueue.shift();
            }

            // Met fin à l'exécution de la boucle for.
            break;
        }

        // Si l'animation courante est terminée.
```

(suite sur la page suivante)

5. pause ou rupture en anglais

(suite de la page précédente)

```
    if (currentAnimation.isFinished)
    {
        // Supprime l'élément courant de la liste.
        animationQueue.splice(i, 1);
    }
}
```

Bien que l'emploi du *switch* soit pratique pour identifier le type d'animation, il ne peut pas être utilisé dans le cas présent car l'instruction *break*; du type *break* doit mettre fin à la *boucle for* et non à l'instruction *switch*.

Toutefois, il est parfois nécessaire d'effectuer des opérations au moment exact où une animation prend fin. Par exemple, lorsque l'animation d'assignation d'une valeur à une variable est terminée, il faut modifier la valeur que contient la variable en question, en accord avec la valeur qui lui a été assignée pendant l'animation. Evidemment, on ne peut pas modifier sa valeur au début ou pendant l'animation, car l'animation perdrait tout son intérêt. C'est pourquoi les animations de type *break* possèdent un attribut *.callback*⁶ qui permet de stocker une fonction dans l'attribut et de l'appeler plus tard comme une fonction normale : *.callback(arguments)*. Ainsi, il suffit de vérifier si une fonction *callback* est stockée dans les animations de type *break* et d'appeler la fonction stockée le cas échéant :

```
update() // Exécutée 60 fois par seconde par Phaser.
{
    // Parcours les animations de la liste (animationQueue).
    for (let i = 0; i < animationQueue.length; i++)
    {
        // Stocke l'animation courante de la liste.
        const currentAnimation = animationQueue[i];

        // Identifie le type de l'animation.
        if (currentAnimation.type == "break")
        {
            // Si le type break est le premier élément de la liste.
            if (i == 0)
            {
                // Retire le premier élément de la liste.
                animationQueue.shift();

                if (currentAnimation.callback != null)
                {
                    // Exécute la fonction callback.
                    currentAnimation.callback();
                }
            }

            // Met fin à l'exécution de la boucle for.
            break;
        }
    }
}
```

6. Fonction de rappel en français, voir glossaire

5.2.2 Déplacement des cartes

Le *framework* Phaser permet de déplacer ses objets par un procédé qui s'appelle le *tweening*⁷. Ce procédé permet en effet de déplacer un objet d'un point A à un point B automatiquement et fluidement. Appliquer du *tweening* sur une carte exécute un code asynchrone modifiant les propriétés *x* et *y* de la carte concernée afin de la faire se diriger vers le point souhaité :

```
// "this" fait référence à la scène de Phaser.
this.tweens.add({
  // Objet concerné par l'animation.
  targets: image,

  // Coordonnées de la destination.
  x: ...,
  y: ...,

  // Durée de l'animation.
  duration: ...,

  // Type d'accélération.
  ease: 'Power1',

  // Fonction appelé à la fin de l'animation
  onComplete: function() {...}
});
```

Cependant, ce principe ne respecte pas le fondement du système d'animation développé précédemment, qui consiste à stocker dans une liste toutes les animations créées, afin de pouvoir les jouer dans un ordre défini, simultanément ou non. En effet, le *tweening* proposé par Phaser déclenche une animation au moment même où le *tweening* est créé, ou éventuellement avec un délai mesuré en microsecondes. Par conséquent, il est préférable que les déplacements des cartes ne relèvent pas de la responsabilité de Phaser.

Il est donc nécessaire que le programme gère ce type d'animation lui-même. Pour cela, l'objet *animation* de type *movement* doit faire appel à des notions de trigonométrie élémentaires afin de calculer l'angle en radians entre le point de départ et le point d'arrivée du déplacement :

```
const animation = {
  // Type de l'animation.
  type: "movement",

  // Autres propriétés de l'animation.
  // ...

  // Angle entre le point de départ et d'arrivé.
  directionAngle: /* Angle en radian */
};
```

Pour obtenir l'angle de la direction, il suffit de calculer l'arc tangente du quotient de la différence d'ordonnée sur la différence d'abscisse entre le point de départ et d'arrivée :

$$\alpha = \arctan\left(\frac{\Delta y}{\Delta x}\right)$$

7. Voir glossaire

Javascript possède nativement un objet *Math*⁸ qui contient une fonction *atan2(y, x)*⁹ qui prend en paramètres les coordonnées *x* et *y* du point d'arrivée relativement au point de départ (0 ; 0) et retourne l'arc tangente formé par le quotient de *y* sur *x*. L'avantage de cette fonction est qu'elle gère elle-même les cas où *x* ou *y* serait égal à 0 :

```
const animation = {
  // Type de l'animation.
  type: "mouvement",

  // Autres propriétés de l'animation.
  // ...

  // Angle entre le point de départ et d'arrivée.
  directionAngle: Math.atan2((animation.y - card.futureY),
                             (animation.x - card.futureX));
```

Comme l'angle de la direction dans laquelle la carte doit se déplacer n'est calculé qu'une seule fois au moment de la création de l'animation, la fonction *update()* de Phaser n'a plus qu'à actualiser les coordonnées de la carte en tenant compte de l'angle. Les fonctions trigonométriques Sinus/Cosinus de l'angle permettent d'obtenir le décalage horizontal et vertical approprié :

```
update() // Exécutée 60 fois par seconde par Phaser.
{
  // Parcours les animations de la liste (animationQueue).
  for (let i = 0; i < this.animationQueue.length; i++)
  {
    // Stocke l'animation courante de la liste.
    const currentAnimation = this.animationQueue[i];

    // Identifie le type de l'animation (mouvement).
    if (currentAnimation.type == "mouvement")
    {
      // Récupère la référence de la carte concernée par l'animation.
      const card = currentAnimation.card;

      // Déplacement de la carte.
      card.x += Math.cos(currentAnimation.directionAngle);
      card.y += Math.sin(currentAnimation.directionAngle);
    }
  }
}
```

5.2.3 Retournement des cartes

Le retournement des cartes permet de montrer ou de cacher à l'utilisateur la valeur d'une carte. Grâce à Phaser, ceci peut être géré aisément. En effet, chaque carte est une instance de la classe *Image*¹⁰ proposée par Phaser. Cette classe possède des propriétés *scaleX* et *scaleY* qui représentent respectivement l'étirement horizontal et vertical de l'image. Ainsi, modifier l'une ou l'autre de ces valeurs, modifie le rendu visuel de la carte en question. La valeur par défaut de ces propriétés est de 1, signifiant un étirement d'échelle 1 : 1 ; plus la valeur est grande, plus l'image est étirée sur l'axe correspondant :

8. https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Math

9. https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Math/atan2

10. <https://photonstorm.github.io/phaser3-docs/Phaser.GameObjects.Image.html>

ScaleX :



Fig. 2 – Effet d'étirement de la propriété scaleX.

Avec cette fonctionnalité, il suffit de rétrécir totalement la carte horizontalement et de l'étirer à nouveau jusqu'à sa taille originelle pour donner l'impression d'un retournement :

```
update() // Exécutée 60 fois par seconde par Phaser.
{
    // Parcours les animations de la liste (animationQueue).
    for (let i = 0; i < this.animationQueue.length; i++)
    {
        // Stocke l'animation courante de la liste.
        const currentAnimation = this.animationQueue[i];

        // Identifie le type de l'animation (retournement).
        if (currentAnimation.type == "flip")
        {
            // Récupère la référence de la carte concernée par l'animation.
            const card = currentAnimation.card;

            card.scaleX -= 0.05;

            // Si la carte a fait un retournement complet.
            if (card.scaleX <= -1)
            {
                // Réinitialise l'étirement à sa valeur d'origine.
                card.scaleX = 1;

                // Animation terminée.
                currentAnimation.isFinished = true;
            }
        }
    }
}
```


6.1 Evolution du projet

Le projet est passé par une multitude de stades et de prototypes avant d'être clair dans ses objectifs et dans sa direction au niveau du développement.

Au commencement, l'outil se voulait être basé sur l'interaction directe de l'utilisateur avec les cartes qui composent la liste. En effet, l'un des premiers prototypes avait comme fonctionnalité primaire le déplacement de cartes à l'aide de la souris. Les interactions directes avec le jeu de cartes (qui représente une liste d'éléments) étaient censées générer une sorte de pseudo-code qui décrit l'équivalent de l'action qui a été effectuée. Par exemple si l'utilisateur avait retourné la troisième carte de la liste, une instruction du type `RetourneCarte(3)` aurait été généré. Cette instruction, stockée dans un historique aurait pu permettre de rejouer les actions effectuées par l'utilisateur. Ainsi, ce dernier aurait pu simuler le comportement d'un algorithme afin de réellement comprendre comment il opère. Réciproquement, l'historique aurait pu contenir à l'avance toutes les actions que performe un algorithme afin de pouvoir rejouer cet algorithme devant l'utilisateur. Le but était de se détacher d'un langage de programmation afin de ne pas être limité par lui dans la compréhension des instructions générées par l'outil.

Finalement, le projet a évolué sur une approche plus personnalisable et plus ouverte au niveau de l'implémentation de l'outil afin de s'adapter aux besoins spécifiques de chaque enseignant. De plus, par son API, dont les fonctions peuvent être appelées directement à partir de code concret (Python, JavaScript ou autres langages), l'outil apporte une vision plus réaliste de la façon dont un algorithme est écrit avec du code réel. Cependant, l'aspect d'interaction entre l'outil et l'utilisateur reste présent, du moment que l'élève a la possibilité d'appeler des fonctions de l'API directement ou indirectement. Ainsi, l'implémentation de l'API dans un programme tiers devrait conserver l'interaction afin de poursuivre la philosophie sur laquelle l'outil a été développé ; il s'agit du constructivisme, une méthode d'apprentissage basé sur la représentation qu'un élève construit avec les interactions qu'il a eues avec un objet.

6.2 Difficultés rencontrées lors du développement

A défaut d'avoir rencontré de nombreuses difficultés techniques, l'accroc principal a été la gestion du temps. En effet, l'estimation du temps pour l'ajout d'une fonctionnalité est très souvent largement inférieure à la réalité car de nombreux petits détails techniques n'apparaissent pas dans l'image globale qu'on s'imagine pour implémenter la fonctionnalité. De plus, programmer exige une grande concentration et d'être totalement impliqué afin d'être productif. Une certaine fatigue mentale survient donc rapidement et des pauses plus ou moins fréquentes sont nécessaires.

Une autre difficulté majeure rencontrée a été la gestion des animations. En effet, il s'agissait de mettre de l'ordre dans la manière dont les animations sont jouées. Par exemple, si une animation est en train d'être jouée et que, par la continuité du code, une autre animation doit être jouée, il faut attendre que la première animation soit finie avant de jouer la deuxième. Mais peut-être que l'on souhaite que les deux animations se jouent simultanément ? Il fallait permettre au développeur d'explicitier parfaitement comment les animations doivent s'enchaîner afin de prévenir tout comportement indéterminé (le code qui joue les animations est asynchrone, une animation qui dépend d'une autre n'a donc aucune connaissance de l'état de cette dernière). Cette difficulté a été surmontée grâce au système d'animation, comme expliqué dans la partie « Fonctionnement de l'outil », notamment grâce au stockage de différents types d'animations dans une liste.

6.3 Scénarios pédagogiques

L'outil est capable de s'adapter à de nombreux cas d'utilisation d'algorithmes sur une liste. Par exemple, des algorithmes simples comme sommer toutes les valeurs contenues dans une liste ou trouver un minimum ou des algorithmes plus complexes comme la recherche dichotomique.

6.4 Pistes d'amélioration

Finalement, l'outil respecte plutôt bien la philosophie du constructivisme et les fonctionnalités de bases sont presque toutes implémentées. Cependant, tout n'a pas pu être implémenté en raison des difficultés évoquées. Il est donc évident que le projet n'a pas atteint son potentiel maximal et diverses pistes d'amélioration sont explorables afin d'étendre les fonctionnalités de l'outil et d'améliorer celles qui sont déjà présentes. Voici par exemple une liste non exhaustive des pistes d'amélioration possibles et les suggestions sur la manière de procéder pour les effectuer :

6.4.1 Variables spéciales

La création de variables spéciales est un élément qui pourrait apporter une meilleure visualisation des fonctions qui accèdent à un élément de la liste. En effet, ces variables spéciales pourraient désigner une variable « de type indice », c'est-à-dire une variable dont le seul but est de servir d'indice pour accéder à un élément d'une liste. Ce type de variable pourrait être représenté d'une couleur différente ou en forme de flèche et pointerait constamment sur l'élément de la liste dont il est l'indice :

Les « cartes » de la liste sont représentées en bleu, les variables « normales » sont en gris et la variable spéciale, ainsi que la carte qu'elle désigne sont en rouge. Ainsi, lors d'un accès à un élément de la liste par une variable spéciale, cette dernière pourrait se manifester (par surbrillance, ou par un déplacement) afin de montrer sa participation à l'opération en cours.

Concrètement, pour implémenter cette fonctionnalité, il faudrait ajouter une nouvelle fonction dans l'API, par exemple `vars.createIndex(name, indexValue)`. Le paramètre `name` correspond au nom de la variable et `indexValue` définit la valeur que contient la variable spéciale (l'indice d'une carte dans la liste). Ensuite, lors de l'appel de la fonction, il suffit de créer une variable similaire aux « normales » mais avec une image différente et leur ajouter un comportement qui modifie la couleur de la carte qu'elles pointent. Idéalement chaque instance de variables spéciales doit être de couleurs différentes afin d'avoir une vision claire de quelle variable spéciale pointe vers

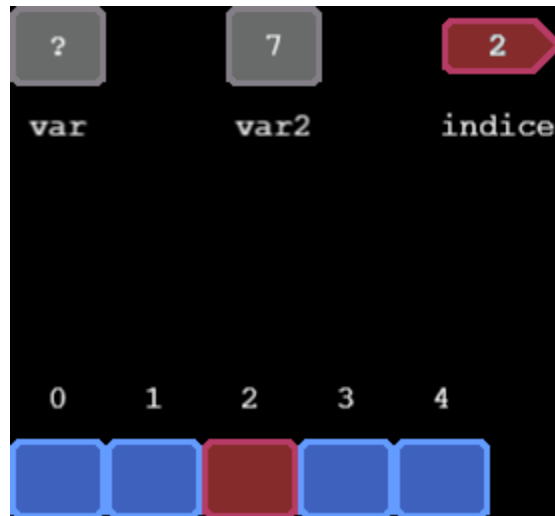


Fig. 1 – Exemple de ce à quoi pourrait ressembler une variable « indice ».

quelle carte. Enfin il suffit de créer une animation qui met simplement en évidence la variable spéciale, et d'exécuter l'animation à chaque fois que celle-ci intervient dans une opération.

6.4.2 Visualisation des conditions

La visualisation des conditions n'est pas implémentée dans l'outil mais permettrait de faciliter la compréhension d'un algorithme.

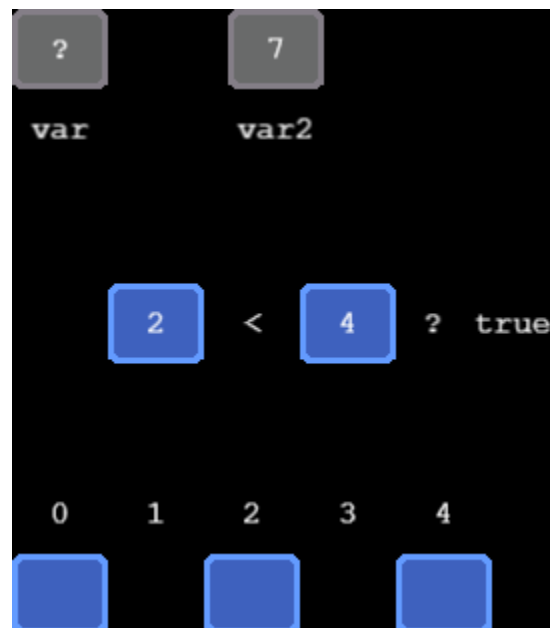


Fig. 2 – Visualisation d'une condition entre la carte 1 et 2.

L'implémentation de cette fonctionnalité nécessiterait de créer une nouvelle animation spécifique qui pourrait être appelée par l'API de cette façon : `cards.compare(index1, index2, comparisonOperator)`. `index1` serait l'indice correspondant à la première carte à comparer, `index2` serait l'indice de la deuxième carte à comparer

et `comparisonOperator` serait l'opérateur de comparaison entre les deux valeurs, par exemple : `==`, `!=`, `<`, `>`, etc...

Quant au développement de l'animation en elle-même, il faudrait la découper en plusieurs séquences isolées et le enchaîner afin de créer une animation fluide. Par exemple, les deux premières cartes pourraient s'élever simultanément afin de repérer quelles sont les cartes qui vont être comparées, puis les retourner en même temps afin de voir leur valeur, suivi de l'apparition de l'opérateur de comparaison entre les cartes et du résultat de la comparaison en dernier temps. Enfin il suffit de faire disparaître l'opérateur et le résultat, de retourner à nouveau les cartes et de les replacer dans la liste. Chaque étape doit évidemment être isolée dans la liste d'animations afin que tout ne se joue pas en même temps.

Afin de tester l'outil, il faut au préalable préparer son environnement :

7.1 Télécharger le dépôt GitHub

Premièrement, il faut se procurer le code du projet. Pour cela, téléchargez le dépôt suivant : <https://github.com/GeinozG/TM-2021>.

7.2 Ouvrir un serveur HTTP local

Il faut à présent configurer un *serveur http* local. Plusieurs manières d'ouvrir un serveur sont possibles mais la plus simple est probablement en utilisant Python. Après avoir téléchargé une version récente de Python, il suffit d'ouvrir une invite de commande dans le dossier du dépôt GitHub précédemment téléchargé et d'entrer la commande suivante : `python -m http.server 8000`. Il ne reste plus qu'à ouvrir un navigateur Web et d'entrer l'adresse suivante : `http://localhost:8000`.

Une zone noire doit normalement être visible sur l'écran. Afin de créer des cartes ou des variables, il faut ouvrir le terminal d'outil de développement (en appuyant sur F12). Depuis cette fenêtre, allez sur l'onglet *console* et entrez ces commandes afin d'initialiser l'API :

```
const cards = new Cards();  
  
const vars = new Variables();
```

7.3 Fonctions disponibles de l'API

7.4 Cartes

Pour utiliser ces fonctions, il suffit d'écrire `cards.` suivi du nom de la fonction et de ses paramètres.

7.4.1 Append

Ajoute un certain nombre de cartes avec des valeurs similaire à la fin de la liste.

```
append(value = 0, nbCards = 1)
```

7.4.2 Append list

Ajoute une liste de cartes avec des valeurs variables à la fin de la liste.

```
appendList(lstValues)
```

7.4.3 Insert

Insère une carte ayant une valeur à un index donné dans la liste.

```
insert(value, index)
```

7.4.4 Pop

Retire le dernier élément de la liste.

```
pop()
```

7.4.5 Swap

Echange la position de deux cartes.

```
swap(index1, index2)
```

7.4.6 Move

Change la position d'une carte par rapport à son index courant et l'index de destination.

```
move(indexCard, indexDestination)
```

7.4.7 Read

Lit la valeur d'une carte et la stocke dans une variable.

```
read(index, outVarName)
```

7.4.8 Assign

Assigne une valeur à une carte de la liste selon son index.

```
assign(index, value)
```

7.4.9 Add

Additionne la valeur de deux cartes et la stocke dans une variable.

```
add(index1, index2, outVarName)
```

7.4.10 Multiply

Multiplie la valeur de deux cartes et la stocke dans une variable.

```
multiply(index1, index2, outVarName)
```

7.4.11 Subtract

Soustrait la valeur de deux cartes et la stocke dans une variable.

```
subtract(index1, index2, outVarName)
```

7.4.12 Divide

Divise la valeur de deux cartes et la stocke dans une variable.

```
divide(index1, index2, outVarName)
```

7.4.13 Show index

Montre ou non l'indice de chaque carte de la liste (vrai par défaut).

```
showIndex(shouldShow)
```

7.5 Variables

Pour utiliser ces fonctions, il suffit d'écrire `vars.` suivi du nom de la fonction et de ses paramètres.

7.5.1 Create

Crée une nouvelle variable.

```
create(name, defaultValue)
```

7.5.2 Delete

Supprime une variable.

```
delete(name)
```

7.5.3 Assign

Assigne une valeur à une variable.

```
assign(name, value)
```

7.5.4 Add

Additionne la valeur actuelle d'une variable à la valeur d'une carte

```
add(varName, cardIndex)
```

7.5.5 Multiply

Multiplie la valeur actuelle d'une variable à la valeur d'une carte

```
multiply(varName, cardIndex)
```

7.5.6 Subtract

Soustrait la valeur actuelle d'une variable à la valeur d'une carte

```
subtract(varName, cardIndex)
```

7.5.7 Divide

Divise la valeur actuelle d'une variable par la valeur d'une carte

```
divide(varName, cardIndex)
```


Framework : « Désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel. » ¹

Canvas : « Ajouté en HTML5, l'élément **canvas** est un nouvel élément qui peut être utilisé pour dessiner des graphismes via des scripts JavaScript. » ²

Tweening : « Le tweening (ou interpolation) est un procédé dans le film d'animation qui permet de créer des images intermédiaires successives de telle sorte qu'une image s'enchaîne agréablement et de façon fluide avec la suivante. » ³

Fonction de rappel : « En informatique, une fonction de rappel (callback en anglais) ou fonction de post-traitement est une fonction qui est passée en argument à une autre fonction. Cette dernière peut alors faire usage de cette fonction de rappel comme de n'importe quelle autre fonction, alors qu'elle ne la connaît pas par avance. » ⁴

1. <https://fr.wikipedia.org/wiki/Framework>

2. https://developer.mozilla.org/fr/docs/Web/API/Canvas_API

3. <https://fr.wikipedia.org/wiki/Tweening>

4. https://fr.wikipedia.org/wiki/Fonction_de_rappel

9.1 Code source

9.1.1 phaser.min.js

Le fichier qui initialise *Phaser* dans le projet est beaucoup trop grand pour être inclus ici. Ce lien permet d'accéder au fichier en question depuis GitHub : « <https://raw.githubusercontent.com/photonstorm/phaser/master/dist/phaser.min.js> »

9.1.2 EventEmitter.js

```
const EventEmitter = new Phaser.Events.EventEmitter();
```

9.1.3 api.js

```
class Cards
{
    append(value = 0, nbCards = 1)
    {
        for (let i = 0; i < nbCards; i++)
        {
            EventEmitter.emit("createCard", value);
        }
    }

    appendList(lstValues)
    {
        EventEmitter.emit("createCards", lstValues);
    }
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

insert(value, index)
{
    eventEmitter.emit("insertCard", value, index);
}

pop()
{
    eventEmitter.emit("popCard");
}

swap(index1, index2)
{
    eventEmitter.emit("swapCard", index1, index2);
}

move(indexCard, indexGo)
{
    eventEmitter.emit("moveCard", indexCard, indexGo);
}

read(index, outVar = "")
{
    eventEmitter.emit("readCard", index, outVar);
}

assign(index, value)
{
    eventEmitter.emit("assignCardValue", index, value);
}

add(index1, index2, outVar = "")
{
    eventEmitter.emit("addCards", index1, index2, outVar)
}

multiply(index1, index2, outVar = "")
{
    eventEmitter.emit("multiplyCards", index1, index2, outVar)
}

subtract(index1, index2, outVar = "")
{
    eventEmitter.emit("subtractCards", index1, index2, outVar)
}

divide(index1, index2, outVar = "")
{
    eventEmitter.emit("divideCards", index1, index2, outVar)
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
showIndex(shouldShow)
{
    eventEmitter.emit("showIndex", shouldShow);
}

class Variables
{
    create(name, defaultValue = '?')
    {
        eventEmitter.emit("createVariable", name, defaultValue);
    }

    delete(name)
    {
        eventEmitter.emit("deleteVariable", name);
    }

    assign(name, value)
    {
        eventEmitter.emit("assignVarValue", name, value);
    }

    add(varName, cardIndex)
    {
        eventEmitter.emit("addToVar", varName, cardIndex);
    }

    subtract(varName, cardIndex)
    {
        eventEmitter.emit("subtractToVar", varName, cardIndex);
    }

    multiply(varName, cardIndex)
    {
        eventEmitter.emit("multiplyToVar", varName, cardIndex);
    }

    divide(varName, cardIndex)
    {
        eventEmitter.emit("divideToVar", varName, cardIndex);
    }
}
```

9.1.4 MainScene.js

```
class MainScene extends Phaser.Scene
{
    // Membres de la scène.
    lstCards;
    lstVariables;

    canvasWidth;
    canvasHeight;

    nbCardsOnSpot;

    animationID;

    animationQueue;

    indexShown;
    lstIndices;

    camera;

    constructor()
    {
        super("MainScene");

        this.lstCards = [];
        this.lstVariables = [];

        this.canvasWidth = 0;
        this.canvasHeight = 0;

        this.nbCardsOnSpot = 0;

        this.animationID = 0;

        this.animationQueue = [];

        this.indexShown = true;
        this.lstIndices = [];
    }

    // ---- Méthodes spécifiques aux scènes de Phaser ----

    preload()
    {
        this.load.image("card", "images/card.png");
        this.load.image("variable", "images/var.png");

        this.canvasWidth = this.sys.game.canvas.width;
        this.canvasHeight = this.sys.game.canvas.height;

        this.camera = this.cameras.main;
    }
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

create()
{
    eventEmitter.on("createCard", this.createCard, this);
    eventEmitter.on("createCards", this.createCards, this);
    eventEmitter.on("insertCard", this.insertCard, this);
    eventEmitter.on("popCard", this.popCard, this);
    eventEmitter.on("swapCard", this.swapCard, this);
    eventEmitter.on("moveCard", this.moveCard, this);
    eventEmitter.on("readCard", this.readCard, this);
    eventEmitter.on("assignCardValue", this.cardAssignment, this);

    eventEmitter.on("addCards", this.addCards, this);
    eventEmitter.on("multiplyCards", this.multiplyCards, this);
    eventEmitter.on("subtractCards", this.subtractCards, this);
    eventEmitter.on("divideCards", this.divideCards, this);

    eventEmitter.on("addToVar", this.addVar, this);
    eventEmitter.on("subtractToVar", this.subtractVar, this);
    eventEmitter.on("multiplyToVar", this.multiplyVar, this);
    eventEmitter.on("divideToVar", this.divideVar, this);

    eventEmitter.on("createVariable", this.createVariable, this);
    eventEmitter.on("deleteVariable", this.deleteVariable, this);
    eventEmitter.on("assignVarValue", this.variableAssignment,
↪this);

    eventEmitter.on("showIndex", this.showIndex, this);
}

update()
{
    for (let i = 0; i < this.animationQueue.length; i++)
    {
        const currentAnimation = this.animationQueue[i];

        if (currentAnimation.type == "move")
        {
            const entity = currentAnimation.entity;

            entity.x += Math.cos(currentAnimation.
↪directionAngle) * 10;
            entity.y += Math.sin(currentAnimation.
↪directionAngle) * 10;

            entity.value.x = entity.x;
            entity.value.y = entity.y;

            if (entity.type == "var")
            {
                entity.name.x = entity.x;
            }
        }
    }
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
entity.name.y = entity.y + entity.height;

    }

    // Quand l'animation est terminée.
    if (Math.abs(entity.x - currentAnimation.x) < 5 && Math.abs(entity.y - currentAnimation.y) < 5)
    {
        entity.x = currentAnimation.x;
        entity.y = currentAnimation.y;
        entity.value.x = entity.x;
        entity.value.y = entity.y;

        if (entity.type == "var")
        {
            entity.name.x = entity.x;
            entity.name.y = entity.y + entity.height;
        }

        this.animationQueue.splice(i, 1);
    }
else if (currentAnimation.type == "flip")
{
    const card = currentAnimation.card;

    card.scaleX -= 0.05;
    // Quand l'animation est terminée.
    if (card.scaleX <= -1)
    {
        if (card.state == "back")
        {
            card.state == "front";
        }
        else if (card.state == "front")
        {
            card.state = "back";
        }

        card.value.visible = !card.value.visible;

        card.scaleX = 1;
        this.animationQueue.splice(i, 1);
    }
}
else if (currentAnimation.type == "spawn")
{
    const card = currentAnimation.card;

    card.visible = true;
    if (card.state == "front")
```

(suite sur la page suivante)

(suite de la page précédente)

```
        {
            card.value.visible = true;
        }

        this.animationQueue.splice(i, 1);
    }
    else if (currentAnimation.type == "value")
    {
        currentAnimation.tempValue.visible = true;
        currentAnimation.tempValue.x += Math.
↪cos(currentAnimation.directionAngle) * 10;
        currentAnimation.tempValue.y += Math.
↪sin(currentAnimation.directionAngle) * 10;

        // Quand l'animation est terminée.
        if (Math.abs(currentAnimation.tempValue.x -
↪currentAnimation.x) < 5 && Math.abs(currentAnimation.tempValue.y -
↪currentAnimation.y) < 5)
        {
            this.animationQueue.splice(i, 1);
            currentAnimation.tempValue.destroy();
        }
    }
    else if (currentAnimation.type == "break")
    {
        if (i == 0)
        {
            this.animationQueue.shift();
            if (currentAnimation.callback != null)
            {
                currentAnimation.callback();
            }
        }

        break;
    }
}

// ---- Méthodes spécifiques aux animations ----

animateSpawn(card)
{
    const animation = {};
    animation.type = "spawn";
    animation.card = card;

    this.animationQueue.push(animation);
}

animateMove(entity, x, y)
{
```

(suite sur la page suivante)

(suite de la page précédente)

```
// Configure l'animation sur la carte.
if (entity.futureX !== x || entity.futureY !== y)
{
    const animation = {};
    animation.type = "move";
    animation.entity = entity;
    animation.x = x;
    animation.y = y;
    animation.directionAngle = Math.atan2((animation.y -
↪entity.futureY), (animation.x - entity.futureX));

    entity.futureX = x;
    entity.futureY = y;

    this.animationQueue.push(animation);
}

animateFlip(card)
{
    const animation = {};
    animation.type = "flip";
    animation.card = card;

    this.animationQueue.push(animation);
}

animateValueAffect(startX, startY, value, entityTo)
{
    const animation = {};
    animation.type = "value";
    animation.entityTo = entityTo;
    animation.x = entityTo.futureX;
    animation.y = entityTo.futureY;
    animation.directionAngle = Math.atan2((entityTo.futureY -
↪startY), (entityTo.futureX - startX));
    animation.tempValue = this.add.text(startX, startY, value).
↪setOrigin(0.5);
    animation.tempValue.visible = false;

    this.animationQueue.push(animation);
}

// Poste un message dans la queue d'animation qui les délimite.
postBreakAnim(callback = null)
{
    const breakAnim = {};
    breakAnim.type = "break";
    breakAnim.callback = callback;

    this.animationQueue.push(breakAnim);
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
// Actualise les coordonnées des cartes par rapport à leur position.
updateCardsCoord(shouldPostBreak = true)
{
    for (let i = 0; i < this.lstCards.length; i++)
    {
        const currentCard = this.lstCards[i];

        this.animateMove(currentCard, i * currentCard.width +
↪ (currentCard.width / 2), this.canvasHeight - (currentCard.height / 2));
    }

    if (shouldPostBreak)
    {
        this.postBreakAnim();
    }
}

// Actualise les coordonnées des variables par rapport à leur
↪ position.
updateVarsCoord(shouldPostBreak = true)
{
    for (let i = 0; i < this.lstVariables.length; i++)
    {
        const currentVar = this.lstVariables[i];

        this.animateMove(currentVar, i * currentVar.width + i
↪ * 60 + (currentVar.width / 2), currentVar.height / 2);
    }

    if (shouldPostBreak)
    {
        this.postBreakAnim();
    }
}

getVarByName(varName)
{
    for (let i = 0; i < this.lstVariables.length; i++)
    {
        const variable = this.lstVariables[i];
        if (variable.name.text === varName)
        {
            return variable;
        }
    }
}

// Affecte une valeur à une variable existante.
assignVarValue(name, value)
{
    let currentVar = this.getVarByName(name);
```

(suite sur la page suivante)

(suite de la page précédente)

```
        currentVar.value.text = value;
    }

    showIndex(shouldShow)
    {
        this.indexShown = shouldShow;
        for (let i = 0; i < this.lstIndices.length; i++)
        {
            this.lstIndices[i].visible = shouldShow;
        }
    }

    // ---- Méthodes appelées par l'API ----

    // Insère une carte à une position spécifique de la liste.
    insertCard(value, index, allInOnePower = true)
    {
        // Crée la carte.
        const card = this.add.image(0, 0, "card").setOrigin(0.5);
        card.type = "card";
        card.value = this.add.text(card.x, card.y, value).setOrigin(0.
→5);

        card.visible = false;
        card.value.visible = false;

        card.x = this.canvasWidth / 2;
        card.y = this.canvasHeight / 2;

        card.value.type = "value";
        card.value.x = card.x;
        card.value.y = card.y;

        card.state = "front"; // ou "back"
        card.futureX = card.x;
        card.futureY = card.y;
        card.value.futureX = card.value.x;
        card.value.futureY = card.value.y;
        card.endTime = 0;

        this.lstCards.splice(index, 0, card);

        const cardIndex = this.add.text(
            48 * this.lstIndices.length + 24,           // x position
            this.canvasHeight - 60,                     // y position
→            this.lstIndices.length,
→            // valeur
        ).setOrigin(0.5)
        cardIndex.visible = false;

        this.lstIndices.push(cardIndex);
    }
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
        this.animateSpawn(card);
        this.updateCardsCoord(allInOnePower);

        if (allInOnePower)
        {
            this.animateFlip(card);
            this.postBreakAnim(() =>
            {
                if (this.indexShown)
                {
                    this.lstIndices[index].visible = this.
→indexShown;
                }
            });
        }

        // Ajoute une carte à la fin de la liste.
        createCard(value)
        {
            this.insertCard(value, this.lstCards.length);
        }

        createCards(lstValues)
        {
            for (let i = 0; i < lstValues.length; i++)
            {
                this.insertCard(lstValues[i], this.lstCards.length,
→false);
            }

            this.postBreakAnim();

            for (let i = 0; i < lstValues.length; i++)
            {
                this.animateFlip(this.lstCards[this.lstCards.length -
→lstValues.length + i]);
            }

            this.postBreakAnim(() =>
            {
                this.showIndex(this.indexShown);
            });
        }

        // Supprime la dernière carte de la liste.
        popCard()
        {
            this.lstCards[this.lstCards.length - 1].value.destroy();
            this.lstCards[this.lstCards.length - 1].destroy();
            this.lstCards.pop();
        }
    }
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

// Echange la position de deux cartes.
swapCard(index1, index2)
{
    const card1 = this.lstCards[index1];
    const card2 = this.lstCards[index2];

    [this.lstCards[index1], this.lstCards[index2]] = [this.
↪lstCards[index2], this.lstCards[index1]];

    this.animateMove(card1, card1.futureX, card1.futureY - 100);
    this.animateMove(card2, card2.futureX, card2.futureY - 100);
    this.postBreakAnim();
    this.updateCardsCoord();
}

// Change l'emplacement de la carte dans la liste.
moveCard(indexInit, indexGo)
{
    const card = this.lstCards[indexInit];

    this.lstCards.splice(indexGo, 0, this.lstCards.
↪splice(indexInit, 1)[0]);

    this.animateMove(card, card.futureX, card.futureY - 100);
    this.postBreakAnim();
    this.updateCardsCoord();
}

// Crée une nouvelle variable.
createVariable(name, value)
{
    const variable = this.add.image(this.canvasWidth / 2, this.
↪canvasHeight / 2, "variable").setOrigin(0.5);
    variable.type = "var";
    variable.name = this.add.text(variable.x, variable.y + ↵
↪variable.height, name).setOrigin(0.5);
    variable.value = this.add.text(variable.x, variable.y, value).
↪setOrigin(0.5);
    variable.value.type = "value";

    variable.futureX = variable.x;
    variable.futureY = variable.y;

    this.lstVariables.push(variable);

    this.updateVarsCoord();
}

// Supprime une variable existante.
deleteVariable(name)
```

(suite sur la page suivante)

(suite de la page précédente)

```
{
    for (let i = 0; i < this.lstVariables.length; i++)
    {
        const currentVar = this.lstVariables[i];

        if (currentVar.name.text === name)
        {
            this.lstVariables[i].value.destroy();
            this.lstVariables[i].name.destroy();
            this.lstVariables[i].destroy();
            this.lstVariables.splice(i, 1);

            break;
        }
    }

    this.updateVarsCoord();
}

// Assigne une valeur à une variable.
variableAssignment(varName, value)
{
    this.animateValueAffect(this.canvasWidth / 2, this.
    ↪ canvasHeight / 2, value, this.getVarByName(varName));
    this.postBreakAnim(() =>
    {
        this.assignVarValue(this.getVarByName(varName).name.
    ↪ text, value);
    });
}

// Additionne 2 cartes et stocke le résultat dans une variable.
addCards(index1, index2, outVar)
{
    const card1 = this.lstCards[index1];
    const card2 = this.lstCards[index2];
    const result = parseInt(card1.value.text) + parseInt(card2.
    ↪ value.text);

    this.animateValueAffect(this.canvasWidth / 2, this.
    ↪ canvasHeight / 2, result, this.getVarByName(outVar));
    this.postBreakAnim(() =>
    {
        this.assignVarValue(this.getVarByName(outVar).name.
    ↪ text, result);
    });
}

// Multiplie 2 cartes et stocke le résultat dans une variable.
multiplyCards(index1, index2, outVar)
{
    const card1 = this.lstCards[index1];
```

(suite sur la page suivante)

(suite de la page précédente)

```
const card2 = this.lstCards[index2];
const result = parseInt(card1.value.text) * parseInt(card2.
↪value.text);

this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, result, this.getVarByName(outVar));
this.postBreakAnim(() =>
{
    this.assignVarValue(this.getVarByName(outVar).name.
↪text, result);
});
}

// Soustrait 2 cartes et stocke le résultat dans une variable.
subtractCards(index1, index2, outVar)
{
    const card1 = this.lstCards[index1];
    const card2 = this.lstCards[index2];
    const result = parseInt(card1.value.text) - parseInt(card2.
↪value.text);

    this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, result, this.getVarByName(outVar));
    this.postBreakAnim(() =>
    {
        this.assignVarValue(this.getVarByName(outVar).name.
↪text, result);
    });
}

// Divise 2 cartes et stocke le résultat dans une variable.
divideCards(index1, index2, outVar)
{
    const card1 = this.lstCards[index1];
    const card2 = this.lstCards[index2];
    const result = parseInt(card1.value.text) / parseInt(card2.
↪value.text);

    this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, parseInt(result), this.getVarByName(outVar));
    this.postBreakAnim(() =>
    {
        this.assignVarValue(this.getVarByName(outVar).name.
↪text, parseInt(result));
    });
}

// Additionne la valeur d'une carte à la valeur actuelle d'une
↪variable.
addVar(varName, cardIndex)
{
    const card = this.lstCards[cardIndex];
```

(suite sur la page suivante)

(suite de la page précédente)

```
const variable = this.getVarByName(varName);
const result = parseInt(card.value.text) + parseInt(variable.
↪value.text);

this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, result, this.getVarByName(varName));
this.postBreakAnim(() =>
{
    this.assignVarValue(this.getVarByName(varName).name.
↪text, result);
});
}

// Soustrait la valeur d'une carte à la valeur actuelle d'une_
↪variable.
subtractVar(varName, cardIndex)
{
    const card = this.lstCards[cardIndex];
    const variable = this.getVarByName(varName);
    const result = parseInt(variable.value.text) - parseInt(card.
↪value.text);

    this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, result, this.getVarByName(varName));
    this.postBreakAnim(() =>
    {
        this.assignVarValue(this.getVarByName(varName).name.
↪text, result);
    });
}

// Multiplie la valeur d'une carte à la valeur actuelle d'une_
↪variable.
multiplyVar(varName, cardIndex)
{
    const card = this.lstCards[cardIndex];
    const variable = this.getVarByName(varName);
    const result = parseInt(variable.value.text) * parseInt(card.
↪value.text);

    this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, result, this.getVarByName(varName));
    this.postBreakAnim(() =>
    {
        this.assignVarValue(this.getVarByName(varName).name.
↪text, result);
    });
}

// Divise la valeur actuelle d'une variable par la valeur d'une carte
divideVar(varName, cardIndex)
{
```

(suite sur la page suivante)

(suite de la page précédente)

```
const card = this.lstCards[cardIndex];
const variable = this.getVarByName(varName);
const result = parseInt(variable.value.text) / parseInt(card.
↪value.text);

this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, parseInt(result), this.getVarByName(varName));
this.postBreakAnim(() =>
{
    this.assignVarValue(this.getVarByName(varName).name.
↪text, parseInt(result));
});
}

// Lis la valeur d'une carte et la stocke en mémoire.
readCard(index, outVarName)
{
    const card = this.lstCards[index];
    const outVar = this.getVarByName(outVarName);

    this.animateMove(card, card.futureX, card.futureY - 100);
    this.postBreakAnim();

    this.animateFlip(card);
    this.postBreakAnim();

    this.animateValueAffect(card.futureX, card.futureY, card.
↪value.text, outVar);
    this.postBreakAnim(() =>
    {
        this.assignVarValue(this.getVarByName(outVarName).
↪name.text, card.value.text);
    });

    this.animateFlip(card);
    this.postBreakAnim();

    this.updateCardsCoord();
    this.postBreakAnim();
}

// Assigne une valeur à une carte.
cardAssignment(index, value)
{
    this.animateFlip(this.lstCards[index]);
    this.postBreakAnim();
    this.animateValueAffect(this.canvasWidth / 2, this.
↪canvasHeight / 2, value, this.lstCards[index]);
    this.postBreakAnim(() =>
    {
        this.lstCards[index].value.text = value;
    });
}
```

(suite sur la page suivante)

(suite de la page précédente)

```
        this.animateFlip(this.lstCards[index]);  
        this.postBreakAnim();  
    }  
}
```

9.1.5 main.js

```
const config = {  
    type: Phaser.AUTO,  
    width: 1000,  
    height: 600,  
    fps: {  
        target: 60,  
        forceSetTimeout: true  
    },  
    scene: [MainScene]  
};  
  
window.onload = function()  
{  
    const game = new Phaser.Game(config);  
}
```


CHAPITRE 10

Déclaration personnelle

Nom : Geinoz

Prénom : Grégoire

Adresse : Chemin de Pré-vert 1, 1630 Bulle

1. Je certifie que le travail : *Visualisation interactive des opérations et algorithmes sur les listes à l'aide d'un jeu de cartes*, a été réalisé par moi conformément au Guide de travail des collèves et aux Lignes directrices de la DICS concernant la réalisation du Travail de maturité.
2. Je prends connaissance que mon travail sera soumis à une vérification de la mention correcte et complète de ses sources, au moyen d'un logiciel de détection de plagiat. Pour assurer ma protection, ce logiciel sera également utilisé pour comparer mon travail avec des travaux écrits remis ultérieurement, afin d'éviter des copies et de protéger mon droit d'auteur. En cas de soupçon d'atteintes à mon droit d'auteur, je donne mon accord à la direction de l'école pour l'utilisation de mon travail comme moyen de preuve.
3. Je m'engage à ne pas rendre public mon travail avant l'évaluation finale.
4. Je m'engage à respecter la Procédure d'archivage des travaux de maturité/travaux interdisciplinaires centrés sur un projet/travaux personnels/travaux de maturité spécialisée (TM/TIP/TP/TMS) en vigueur dans mon école.
5. J'autorise la consultation de mon travail par des tierces personnes à des fins pédagogiques et/ou d'information interne à l'école :

☐ oui

☐ non (car il contient des données personnelles et sensibles.)

Lieu, date : Bulle, le 3 avril 2022

Signature : Grégoire Geinoz

C

Canvas:, [35](#)

F

Fonction de rappel:, [35](#)

Framework:, [35](#)

T

Tweening:, [35](#)