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

# Développement d'un outil informatique démontrant visuellement la perception d'une liste pour un ordinateur

*Collège du sud, Travail de maturité Version intermédiaire*

**Grégoire Geinoz**

**janv. 12, 2022**

# Table des matières

CHAPITRE 1

# Introduction

## 1.1 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 clair 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 compose une carte avec sa valeur affichée d'un côté de la carte. Ainsi, tous les éléments de la liste forment à eux, 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 API(à expliquer) et sont représentées dans une zone de l'écran spécifique. Cependant, des variables "spéciales", qui n'ont comme seul rôle, d'être un index qui parcours les éléments d'une liste (dans une "boucle for" par exemple), peuvent également être créées et sont représentées comme des flèches qui pointent sur l'élément de la liste qui correspond à l'index 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, il fera discrétement appel à l'API afin qu'une animation se déclenche et montre sur l'écran le comportement de la liste.

## 1.2 Intêret 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 en posant brique par brique des blocs de code afin d'amener une cible jusqu'à son objectif. Le projet envisagé dans le cadre de ce travail de maturité cible un public plus mature capable de pousser un raisonnement plus en profondeur, en effet les listes sont une notion plus abstraite car

elles sont des conteneurs de valeurs quelconques. De plus, ce projet vise à enseigner des notions très spécifiques de la programmation : les listes.

## 1.3 Technologies utilisées

Le projet repose sur 2 technologies principales : Javascript et HTML5. Cependant pour alléger la quantité de travail, un framework de jeu 2D, Phaser, 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.

## 1.4 Configuration matérielle requise pour utiliser l'outil

Afin que l'utilisation de l'outil soit la plus accessible possible, les langages de programmation du Web ont été utilisé pour le développer. Ce qui signifie que pour utiliser l'outil, seul un accès à un ordinateur opérationnel étant doté d'un navigateur internet, et d'une connexion, suffit.

## 1.5 Connaissance requises pour utiliser l'outil

L'outil fait appel aux notions de variables ; éléments qui constituent la liste, ainsi qu'aux opérations de bases entre variables numérique (calculs, affectations de valeurs) et la notion de condition entre les variables utilisant des opérateurs de conditions.

## 1.6 Connaissances requises pour comprendre le fonctionnement de l'outil

Afin de comprendre comment l'outil fonctionne, il est nécessaire de comprendre le code qui le compose. C'est à dire qu'il faut avoir un niveau de base qui couvre tous les fondamentaux du Javascript. Le code étant largement commenté, il n'est pas forcément nécessaire de savoir utiliser le framework Phaser pour comprendre le code source.

CHAPITRE 2

# Aspects pédagogiques

CHAPITRE 3

# API

CHAPITRE 4

# Scénarios d’utilisation

CHAPITRE 5

# Fonctionnement du projet

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

## 5.1 L'utilisation de Phaser

### 5.1.1 Système de gestion des cartes

### 5.1.2 Système d'évènements

### 5.1.3 Gestion de la scène et de la boucle principale

## 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 un attribut « animation » qui possède les informations nécessaire à décrire une animation, par exemple, de déplacement :

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

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

(suite sur la page suivante)

(suite de la page précédente)

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

Et qu'ainsi, dans la fonction « update » gérée par Phaser, une « boucle for » parcours toutes les cartes et effectue l'animation qui lui est attachée, si celle-ci est a animée :

```
update()
{
    // Parcours 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 procédé comporte un gros désavantage. En effet, elle ne laise au programme que la possibilité de gérer toutes les animations en même temps, ce qui signifie que si le développeur souhaite jouer des animations dans un certain ordre, il doit attendre que l'animation précédente soit terminer avant de configurer l'animation 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 un attribut « 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)
{
    const animation = {
        // Type de l'animation.
        type: "movement",

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

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

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

### 5.2.2 Déplacements des cartes

CHAPITRE 6

# Indices and tables

— genindex
— search