Fiche

I. Modifier un algorithme pour répondre au besoin

Un **algorithme** décrit, étape par étape, la manière de résoudre un problème. Avant d'écrire du code, on **adapte** l'algorithme au **besoin** exprimé par le **cahier des charges**.

Exemple de la borne d'arcade
Un des objectifs lorsque l'on allume la borne d'arcade peut être d'avoir un écran d'accueil qui n'affiche que les jeux installés, compatibles avec une manette que l'utilisateur vient de brancher et, éventuellement, marqués comme des jeux favoris, le tout en évitant les doublons.
On formalise les **règles logiques** :

• **ET (AND)** : jeu installé ET compatible manette → jeu visible ;

• **OU (OR)** : filtrer par type de jeu (combat, course, etc.) → deux catalogues réunis ;

• **NON (NOT)** : si un jeu en double → exclure les doublons.

Exemple d'algorithme (pseudo-code) :

1. **Charger** la liste des jeux (titre, genre).

2. **Retirer** ceux dont le fichier est manquant (NON installés).

3. **Garder** les jeux **compatibles** avec la manette branchée (ET).

4. Si le filtre « Favoris » est actif : ne garder QUE les jeux favoris = vrai.

5. **Trier** (titre, puis genre).

6. **Afficher** la liste et attendre un événement (joystick, bouton, timer).

On anticipe aussi des cas limites : fichiers renommés, affiche du jeu absente, manette débranchée. L'algorithme doit prévoir des **chemins de secours** : afficher une icône par défaut, proposer le clavier si la manette est absente, ignorer un jeu corrompu...

II. Traduire l'algorithme en programme

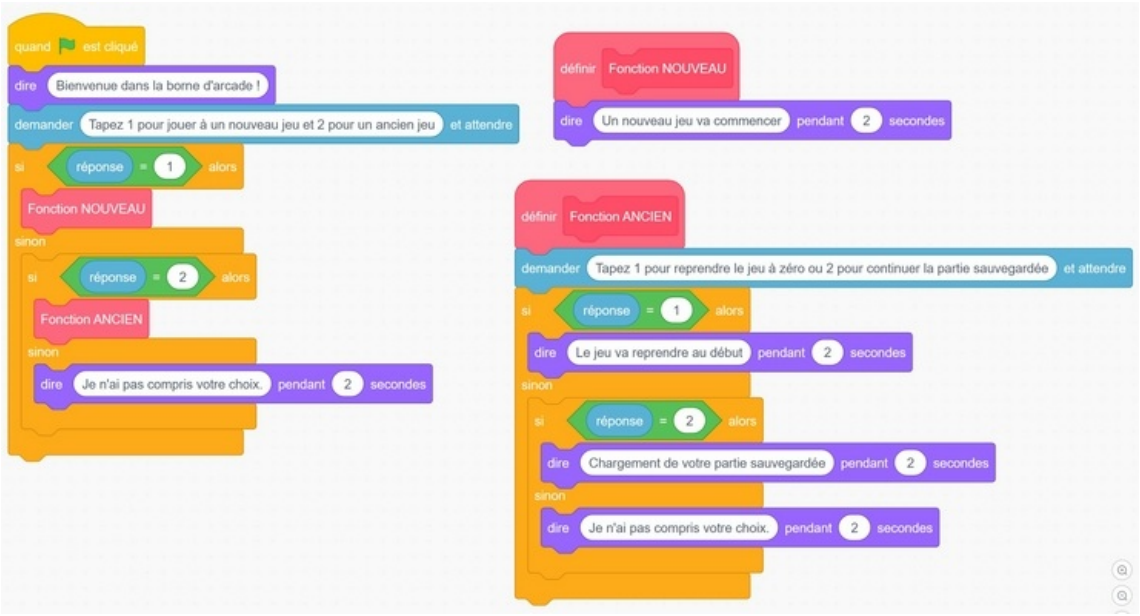
Traduire, c'est passer du **raisonnement** à une **implémentation concrète** (langage, fichiers de configuration, scripts). On identifie les **données** (listes de jeux, menus), les **traitements** (filtrer, trier, lancer un jeu) et les **événements** (actions de l'utilisateur, fin d'un processus).
Cette traduction mène à un programme pouvant être écrit dans différents langages (Scratch, Python, C++). En quatrième, on utilise le langage **Scratch**.
Un **programme informatique** est une **suite d'instructions logiques** que l'ordinateur exécute pour répondre à un besoin ou résoudre un problème. Il suit un ordre précis, comme une recette de cuisine : chaque étape dépend de la précédente.
Un programme bien conçu se compose souvent de plusieurs parties qui communiquent entre elles. Pour éviter de répéter du code, on regroupe certaines instructions dans des fonctions (ou sous-programmes). Une fonction est un morceau de programme réutilisable qui réalise une action précise (par exemple, afficher un message de bienvenue ou calculer un score). Les fonctions permettent d'écrire un code plus clair, plus court et plus facile à corriger. Chaque fonction a un nom, des paramètres (les données qu'elle reçoit), et parfois une valeur de retour.
Ainsi, la traduction d'un algorithme en programme consiste à découper les tâches en blocs cohérents, chacun ayant une fonction claire, pour former un tout **structuré** et **fonctionnel**.

Exemple de la borne d'arcade, en langage Scratch
Prenons maintenant un exemple en langage Scratch, adapté au projet de borne d'arcade.
Lorsque la borne s'allume, le programme doit afficher un menu demandant au joueur ce qu'il souhaite faire :

• jouer à un nouveau jeu ;

• reprendre un ancien jeu.

Si le joueur choisit un ancien jeu, un deuxième choix est proposé : reprendre la partie à zéro, ou charger une sauvegarde et poursuivre le jeu à l'endroit où il s'était arrêté la fois précédente. Cet enchaînement de choix peut être représenté en blocs Scratch sous forme de fonctions :



Cet exemple de programme Scratch montre les principales notions autour du programme informatique que l'on doit connaître en classe de quatrième :

• la **structure séquentielle** (l'ordre des instructions) ;

• les **conditions logiques** (si... sinon) ;

• **l'appel de fonctions** (sous-programmes) comme « Menu Ancien jeu » ou « Menu Nouveau jeu » ;

• **la gestion des réponses** de l'utilisateur (interface homme-machine).

Dans l'ordinateur d'une borne d'arcade réelle, ces blocs Scratch pourraient être remplacés par un code informatique en langage Python ou C++ exécuté sur le PC de la borne, tout en suivant la même logique d'algorithmes et de fonctions.

III. Réaliser et mettre au point un programme commandant un système réel (IHM)

Mettre au point un programme consiste à **modifier** l'algorithme, **adapter** le code, tester pas-à-pas (simulation) puis **tester** en vrai (sur la borne). **L'interaction homme-machine** (IHM) doit rester fluide et prévisible.

Exemple de la borne d'arcade
Au bout de deux minutes d'inactivité, la borne passe en mode « attractif » (diaporama de jaquettes et musique douce d'attente). À la moindre action (joystick/bouton), la borne retourne immédiatement au menu de choix.
Mise au point du programme :

1. Modifier l'algorithme : ajouter un timer_inactivité, un état Attractif (attract_mode, démo d'attente) et les transitions :

• ETAT Menu → (si inactivité ≥ 120 s) → ETAT Attractif.

• ETAT Attractif → (si événement utilisateur) → ETAT Menu.

2. Adapter le programme : créer un gestionnaire d'événements qui réinitialise timer_inactivité à chaque entrée, et une boucle qui déclenche l'attract_mode.

3. Tester en simulation (pas-à-pas) :

• Forcer inactivité = 119 s → rester en menu.

• Passer à 120 s → basculer en attractif.

• Simuler bouton_start → retour menu.

4. Tester sur l'objet réel :

• Mesurer la latence entre l'appui sur le bouton et l'action visible (objectif < 100 ms).

• Vérifier le volume audio en attractif, l'interruption immédiate au premier input.

• Tester les cas limites : manette débranchée en attractif, jeux manquants au retour menu.

À retenir :

1. **Modifier un algorithme**, c'est clarifier les **conditions logiques** (ET/OU/NON), les données utiles et les cas limites.

2. **Traduire en programme**, c'est structurer les **données, traitements, événements** (fichiers JSON, scripts, commandes d'émulateurs).

3. **Mettre au point** passe par la **simulation pas-à-pas**, la **journalisation**, puis le **test sur matériel réel** avec des critères mesurables (latence, stabilité, robustesse).

Définitions importantes :

- **Opérateurs logiques** : ET (toutes vraies), OU (au moins une vraie), NON (négation).
- **Événement** : action/signal qui déclenche une séquence (appui sur un bouton, timer, fin d'un processus).
- **IHM (interface homme-machine)** : ensemble des moyens d'interaction (écran, sons, contrôles).
- **Émulateur** : logiciel qui reproduit le fonctionnement d'un jeu vidéo d'origine.
- ***Front-end* (lanceur)** : interface qui présente les jeux et lance l'émulateur approprié.
- **Journalisation (*logs*)** : enregistrement des événements pour analyser et corriger le fonctionnement.