Bloc électronique

Livrable 3 : Algo 3000

Groupe 10 :

GERVILLIERS Maxence

BAL Aboubacry

LESUEUR Enzo

Martin Léo

RAKOTONIRINA Tony

VAUDRY Garance

Introduction :

Récemment, nous avons réalisés les montages électroniques permettant à une carte d'agent activer le coffre-fort et identifier le modèle de carte insérée.

La séquence 7 nous a permis de valider un système d'authentification par le code agence saisi sur des boutons poussoirs.

Il nous reste pour ce livrable 3 à concevoir l'algorithme du système interne d'authentification, celui-ci se compose de différent type de mécanismes d'authentifications :

MA1 : authentification sous forme de questions / réponses

MA2 : authentification par un code qui change au cours du temps

MA3 : authentification par scan rétinien

MA4 : authentification par scan digital

MA5 : authentification par CARD_ID

Chaque modèle de carte est associé à un niveau de sécurité.

Chaque niveau de sécurité propose une combinaison différente de mécanisme d'authentification :

Niveau de sécurité 1 : MA1 + MA3

Niveau de sécurité 2 : MA1 + MA4

Niveau de sécurité 3 : MA2 + MA5

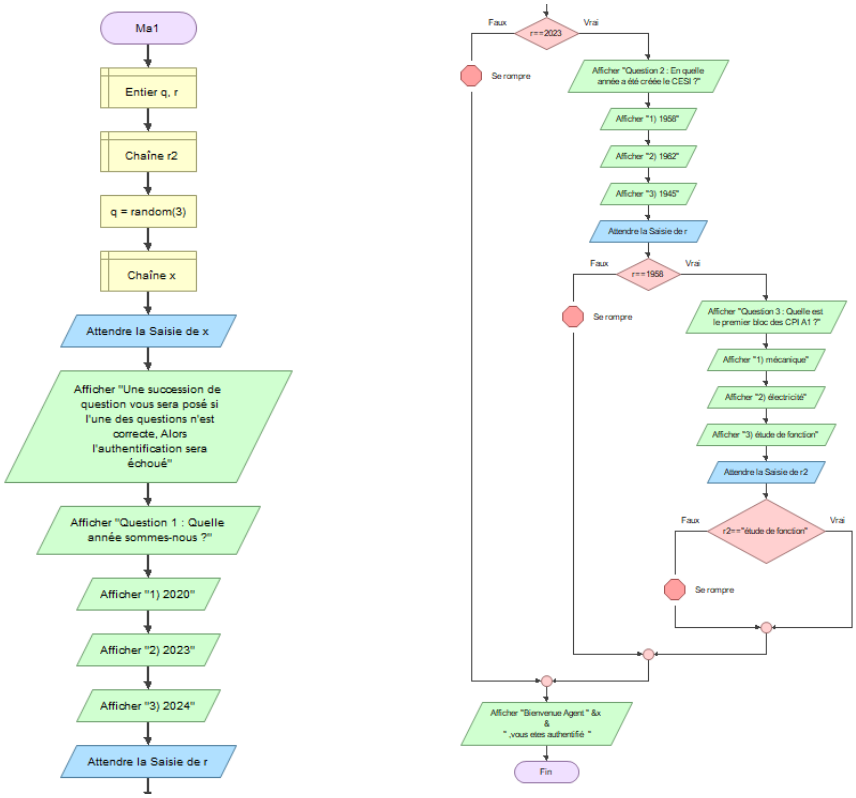
Niveau de sécurité 4 : MA2 + MA3 + MA4

Niveau de sécurité 5 : MA1 + MA2 + MA3 + MA5

Plan D'action :

- Algorithme de chaque mécanisme d'authentification (en logigramme)
- Algorithme principal du système d'authentification du coffre décrivant le processus complet (en logigramme)
- Liste de variable utilisé avec leur type (+ leur porté)

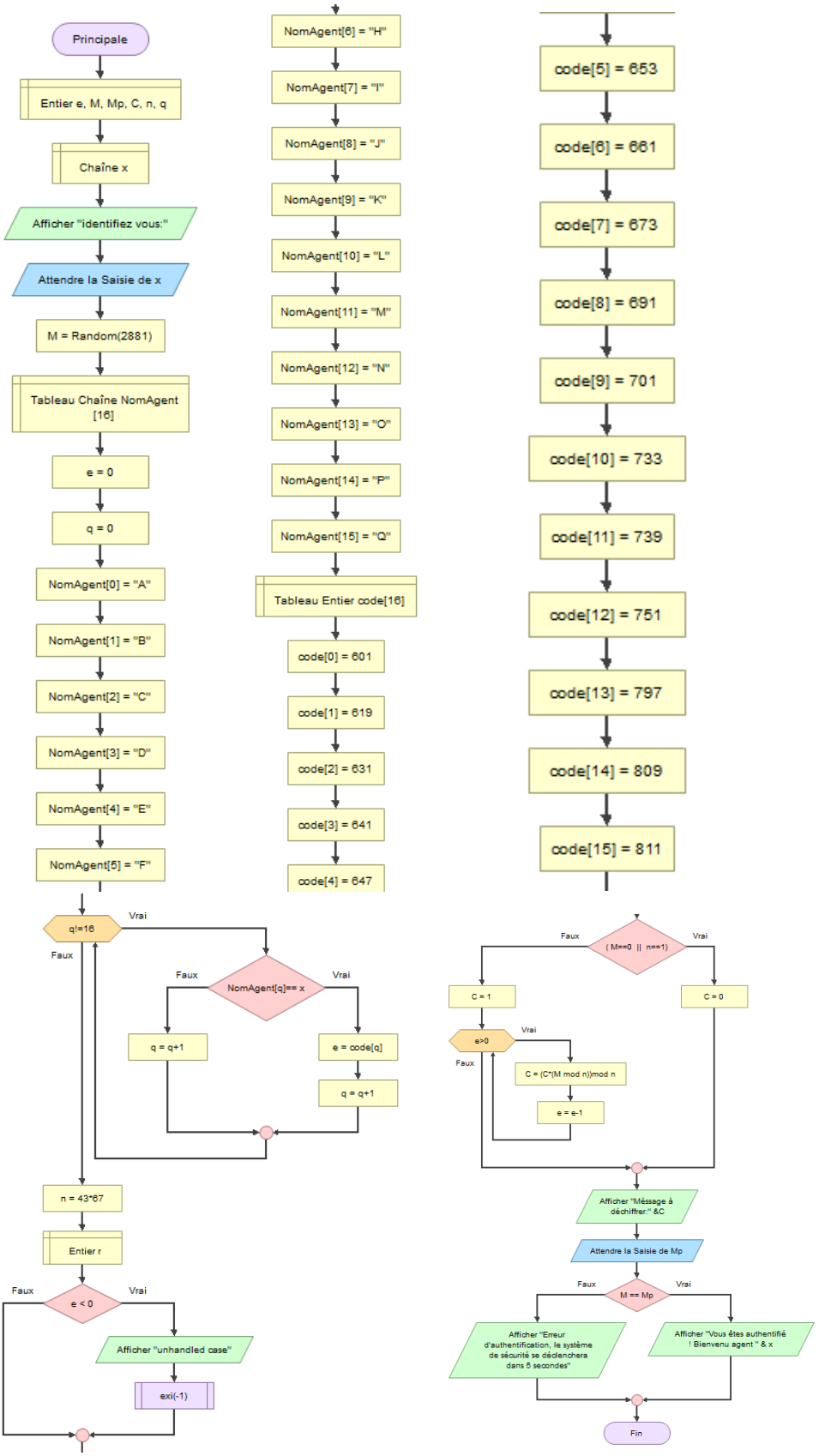
Dans un premier temps, sera présenté les algorithmes de chaque mécanisme d'authentification, ainsi, voici l'algorithme MA1 portant sur des questions réponses :



Cet algorithme vient poser à l'agent une multitude de questions afin de l'authentifier. La questions vient s'afficher en proposant 3 réponses.

Si l'agent renseigne la bonne réponse, alors il pose la questions suivante, jusqu'à le questionnaire réussi. Sinon, si l'agent ne trouve pas la bonne réponses, l'algorithme vient se rompre.

Voici le deuxième mécanisme d'authentifications :



Le code commence par initialiser un générateur de nombres aléatoires en utilisant srand(time(0)), ce qui permet de générer un nombre aléatoire m basé sur l'heure actuelle.

L'utilisateur est invité à s'identifier en entrant son nom d'agent, stocké dans la variable x.

En fonction de la valeur de x, une valeur numérique e est attribuée en utilisant une série de conditions if. Chaque nom d'agent correspond à une valeur spécifique de e.

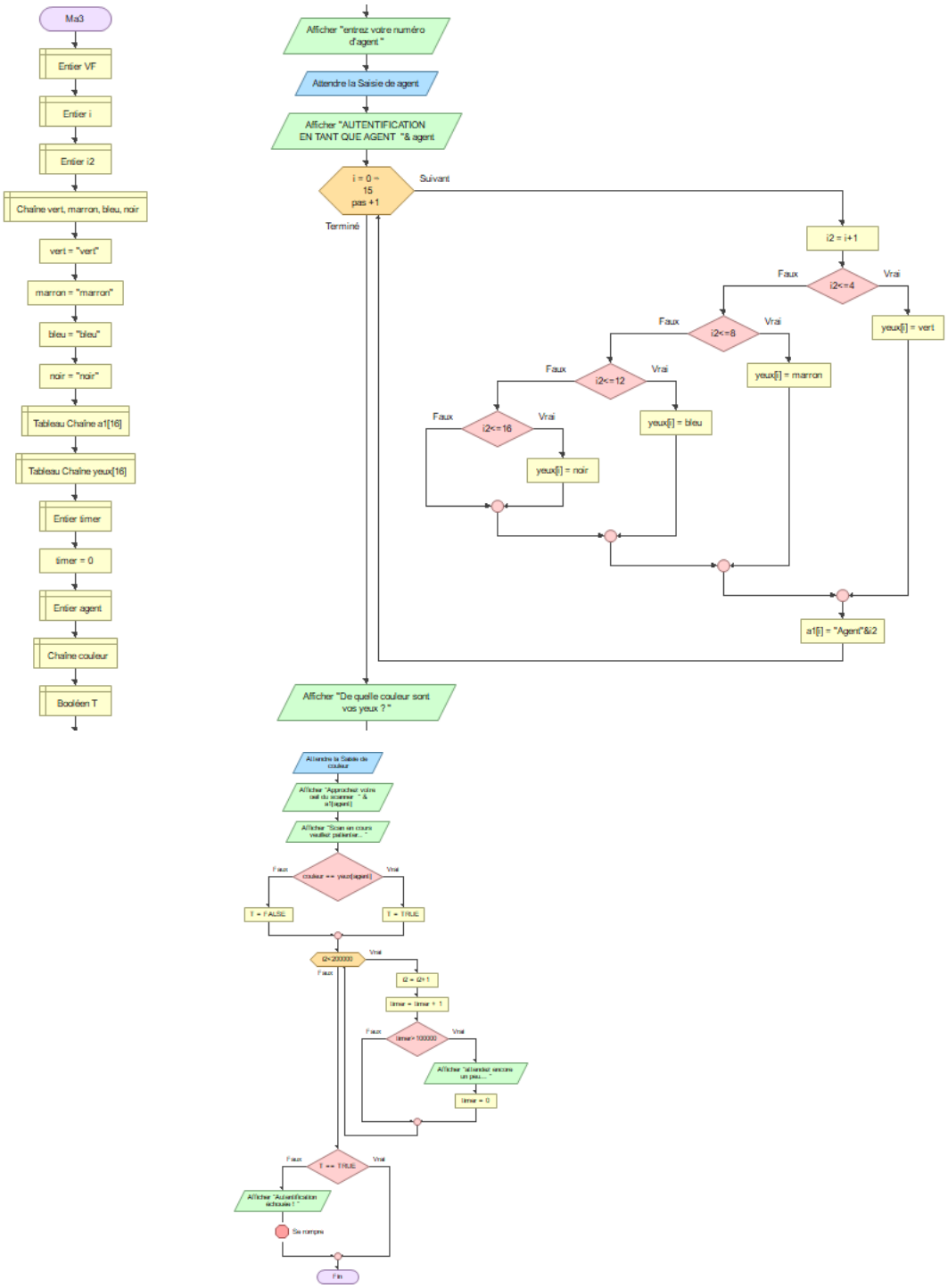
Le code effectue ensuite un calcul mathématique basé sur e, m, et une constante n (le produit de 43 et 67). Ce calcul est utilisé pour générer une valeur c.

L'utilisateur est invité à entrer une valeur mp.

Le code vérifie si m est égal à mp, et si c'est le cas, il affiche un message d'authentification réussie en indiquant le nom de l'agent. Sinon, il affiche un message d'erreur d'authentification.

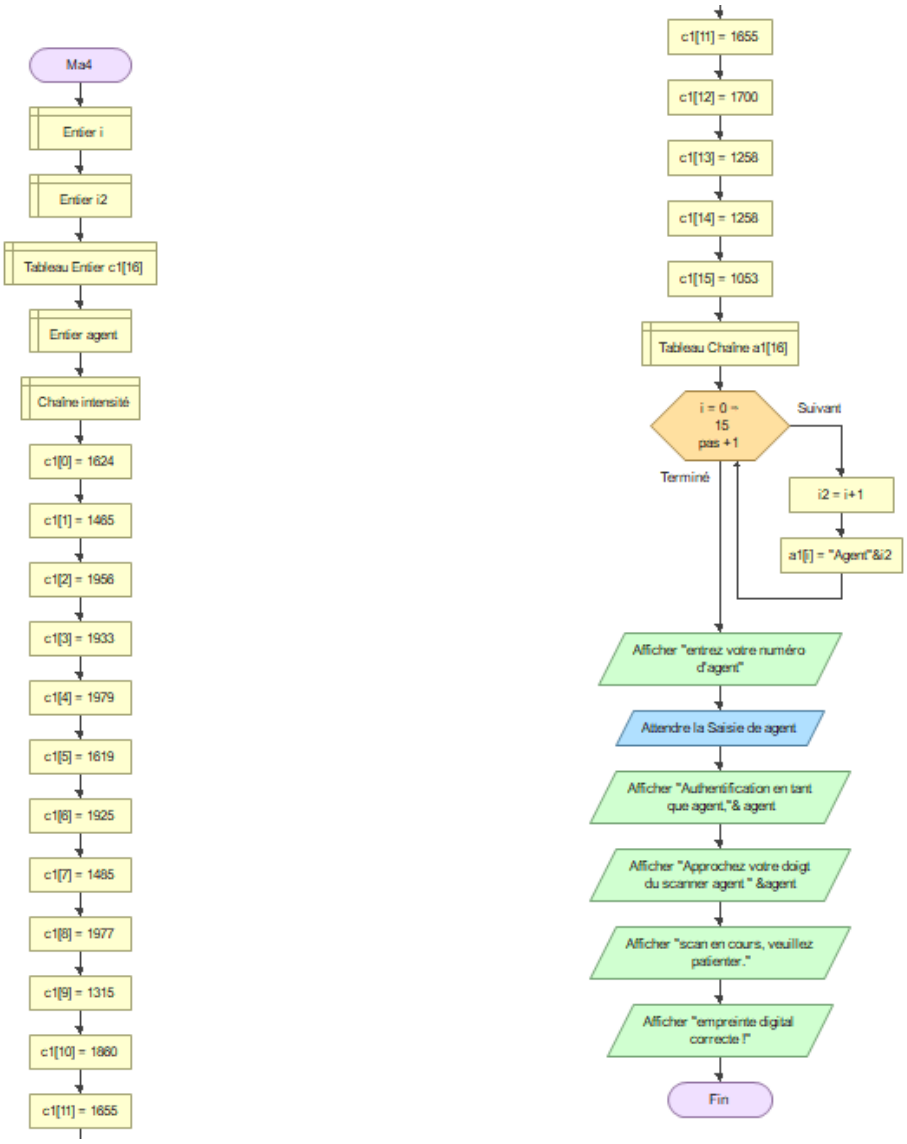
En résumé, ce code semble simuler un processus d'authentification où l'utilisateur doit entrer son nom d'agent, puis des calculs mathématiques basés sur ce nom d'agent sont effectués pour générer une valeur de validation. Si la valeur entrée par l'utilisateur correspond à cette valeur de validation, l'utilisateur est authentifié. Sinon, une erreur d'authentification est signalée.

Ensuite, nous avons le troisième mécanisme d'authentification :



Cet algorithme permet d'identifier l'agent grâce à un scan rétinien. Pour ce faire, nous avons dans un premier temps défini la couleur des yeux de chaque agents(1->4vert ;5->8marron ;9->13 marron ;14->16 noir), puis nous avons fait en sorte de demander à l'agent son numéro d'agent ainsi que la couleur de ses yeux ; si cela correspond au tableau établi, alors l'authentification sera réussie.

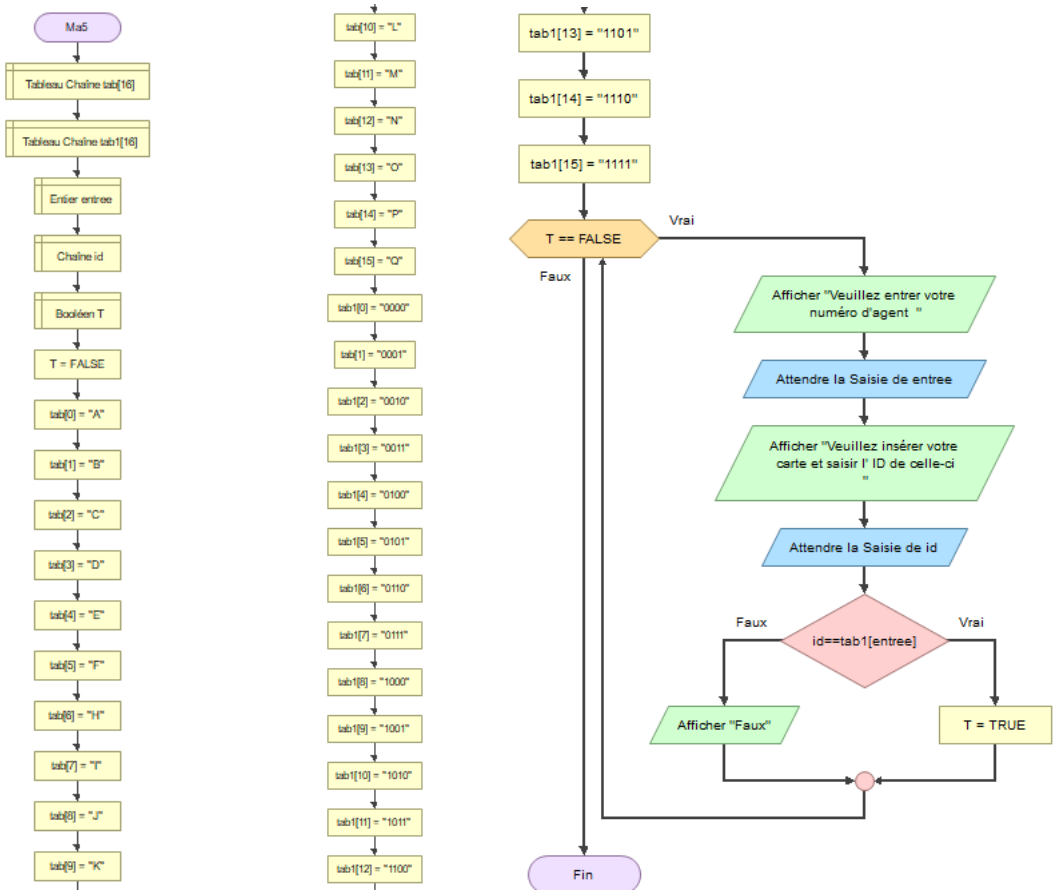
Ensuite voici le quatrième mécanisme d'authentification :



Cet algorithme permet d'identifier l'agent à l'aide d'un scanneur d'empreinte digitale. Pour ce faire nous avons dans un premier temps défini un seuil de tension électrique (entre 1000 et 2000 Ohm propre à chaque agent, afin de déterminer s'il correspond bien à leur numéro d'agent. (nous ne pouvons pas simuler réellement la détection de de la tension traversant le doigt de l'individu testé, donc on suppose qu'il sera bel et bien l'agent) .

On a dans un premier temps ajouté un « if » pour ne laisser rentrer que les 16 agents, de telle sorte à, ce qu'ils puissent rentrer lors qu'ils entrent un nombre inférieur ou égal à 16. On a ensuite simulé, sous-entendu que l'authentification était correcte, et laissé l'accès à l'agent.

Et dernièrement, le cinquième mécanisme d'authentification :



• Les variables tab et tab1 sont des listes de chaînes de caractères (chaine) de longueur 16. tab est initialisé avec des lettres des agents (A à Q), tandis que tab1 est initialisé avec des chaînes de 4 chiffres binaires (0000 à 1111), les id des cartes.

• La variable t est initialisé à False. Elle sera utilisée pour contrôler la boucle while.

• Une boucle while est utilisée pour demander à l'utilisateur de saisir son numéro d'agent et l'ID de sa carte. La boucle continue jusqu'à ce que t devienne True.

• À chaque itération de la boucle, l'utilisateur est invité à entrer son numéro d'agent (entrée) en utilisant entier (input) pour lire un entier à partir de l'entrée standard.

• Ensuite, l'utilisateur est invité à saisir l'ID de sa carte (id) en utilisant input (chaine).

• Le code vérifie si l'ID de la carte (id) correspond à l'ID stocké dans tab1 à l'index spécifié par le numéro d'agent (entrée). Si c'est le cas, la variable t est définie sur True, ce qui fait sortir de la boucle while.

• Si l'ID de la carte ne correspond pas, un message "Faux" est affiché, et la boucle continue à demander à l'utilisateur de saisir son numéro d'agent et l'ID de sa carte.

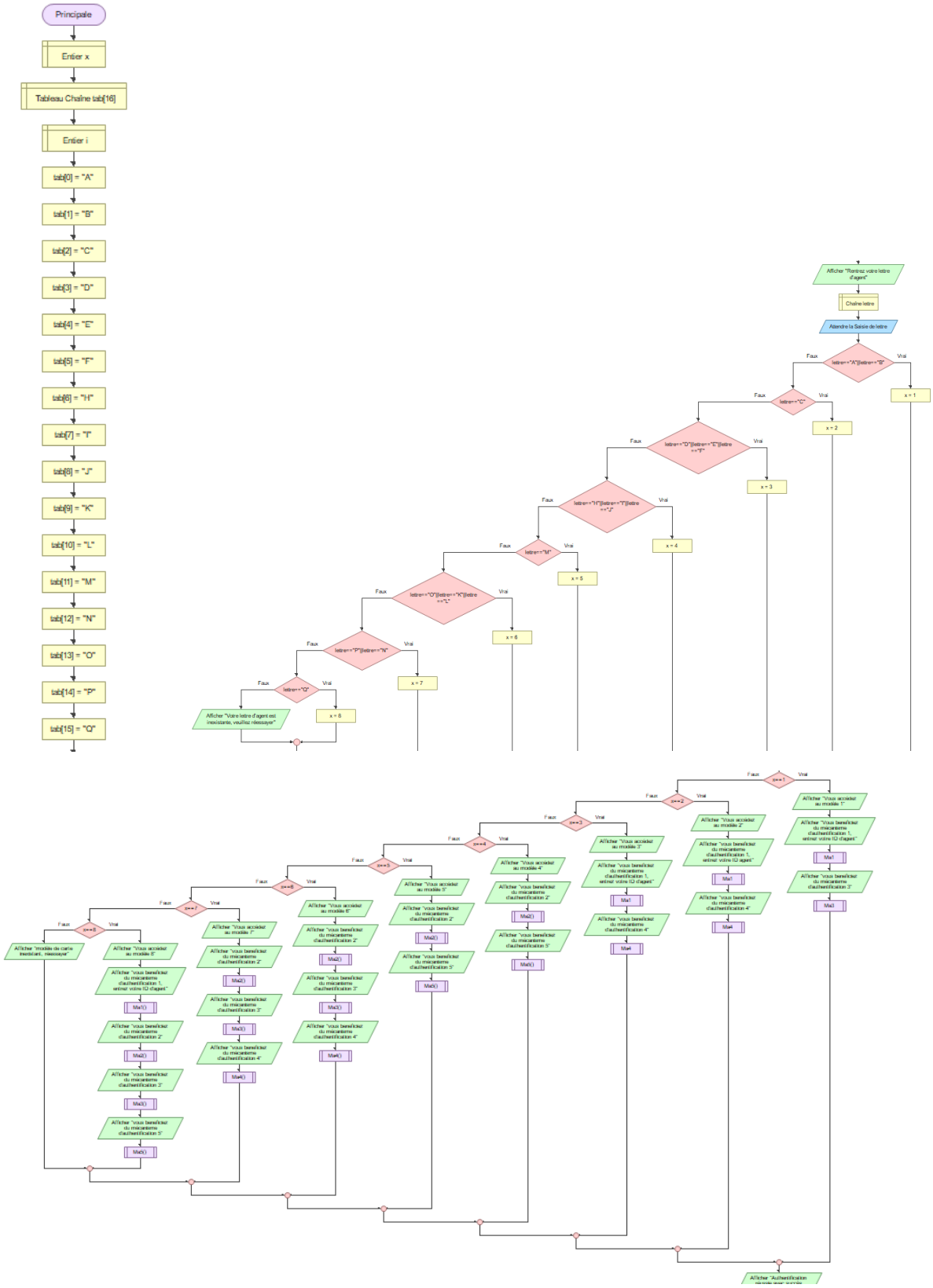
• Une fois que la boucle while est sortie (c'est-à-dire lorsque t est True), le message "Succès" est affiché, indiquant que l'authentification a réussi.

• tab, tab1, et t sont des variables globales qui sont accessibles dans toute la fonction, tandis que entrée et id sont des variables locales qui sont limitées à la portée de la boucle while. Une fois que la boucle while se termine, les variables entrée et id ne sont plus accessibles.

Une fois tous les algorithme de mécanisme d'authentification réalisés, nous avons les modèle de chaque carte qui appartiennent chacun à un niveau de sécurité, pour ce faire, nous avons choisi de donner tel niveau de sécurité à tel modèle de carte. Une fois les choix faits, nous les avons rangés dans un tableau :

niveau 1	modèle 1
niveau 2	modèle 2 et modèle 3
niveau 3	modèle 4 et modèle 5
niveau 4	modèle 6 et modèle 7
niveau 5	modèle 8

Cela fait, voici l'algorithme principe comprenant demandant de rentrer sa lettre de carte et cet algorithme vient alors faire passer à l'agent les mécanisme d'authentification correspondant :



Dans cet Algorithme, dans un premier temps on vient nous demander dans un premier temps la lettre de d'agent et l'algorithme va ranger l'agent dans son modèle de carte correspondant et grâce à ce modèle, il fera passer ensuite à l'agent un niveau de sécurité. En répondant correctement au niveau de sécurité l'agent sera authentifié.

Pour finir, nous avons énumérer dans un tableaux toute les variables utilisées, ainsi que leurs utilités, leurs types et leur porté.

Dans un premier temps, pour MA1 :

Variable	Utilité	Type	Portée
q	Stocke un nombre aléatoire entre 0 et 2	int	Locale à main
r	Stocke les réponses aux questions	int	Locale à main
r2	Stocke la réponse à une question	string	Locale à main
x	Stocke le nom de l'agent	string	Locale à main
vF	Variable inutilisée	int	Locale à main
i, i2	Variables de boucle	int	Locale à main
vert, marron, bleu, noir	Couleurs pour les yeux	string	Locale à main
a1	Stocke les noms d'agents	tableau de string	Locale à main
yeux	Stocke les couleurs des yeux pour chaque agent	tableau de string	Locale à main
timer	Compte le temps dans une boucle	int	Locale à main
agent	Stocke le numéro de l'agent	int	Locale à main
agent	Stocke le numéro de l'agent	int	Locale à main
couleur	Stocke la couleur des yeux saisie par l'utilisateur	string	Locale à main
t	Stocke le résultat de la comparaison des couleurs des yeux	bool	Locale à main
nS1 à nS5	Fonctions déclenchées en fonction des étapes du programme	fonctions booléennes	Locale à main

Ensuite pour MA2 :

Variable	Utilité	Type	Portée
e	Stocke un code d'authentification	int	Locale à `main`
m	Stocke un nombre généré aléatoirement	int	Locale à `main`
mp	Stocke un nombre entré par l'utilisateur	int	Locale à `main`
c	Stocke le résultat du déchiffrement	int	Locale à `main`
n	Stocke un produit de deux nombres	int	Locale à `main`
q	Variable d'itération pour un tableau	int	Locale à `main`
x	Stocke le nom de l'agent entré par l'utilisateur	string	Locale à `main`
nomAgent	Tableau de chaînes de caractères pour stocker des noms d'agents	string array	Locale à `main`
code	Tableau d'entiers pour stocker des codes d'authentification	int array	Locale à `main`
r	Variable non utilisée	int	Locale à `main`

Puis pour MA3 :

vF	Variable non utilisée	int	Locale à `main`
i	Variable utilisée pour les boucles	int	Locale à `main`
i2	Variable utilisée pour les boucles	int	Locale à `main`
vert	Variable contenant une couleur des yeux	string	Locale à `main`
marron	Variable contenant une couleur des yeux	string	Locale à `main`
bleu	Variable contenant une couleur des yeux	string	Locale à `main`
noir	Variable contenant une couleur des yeux	string	Locale à `main`
a1	Tableau de chaînes de caractères pour stocker des noms d'agents	string array	Locale à `main`
yeux	Tableau de chaînes de caractères pour stocker des couleurs des yeux	string array	Locale à `main`
timer	Variable utilisée pour le suivi du temps	int	Locale à `main`
agent	Variable pour stocker le numéro d'agent entré par l'utilisateur	int	Locale à `main`
couleur	Variable pour stocker la couleur des yeux entrée par l'utilisateur	string	Locale à `main`
t	Variable pour stocker le résultat de l'authentification	bool	Locale à `main`

Quatrièmement pour MA4 :

i	Variable utilisée pour les boucles	int	Locale à main
i2	Variable utilisée pour les boucles	int	Locale à main
c1	Tableau d'entiers pour stocker des valeurs d'intensité	int array	Locale à main
agent	Variable pour stocker le numéro d'agent entré par l'utilisateur	int	Locale à main
intensité	Variable pour stocker l'intensité (incohérent avec le type)	string	Locale à main
a1	Tableau de chaînes de caractères pour stocker des noms d'agents	string array	Locale à main
timer	Variable utilisée pour le suivi du temps	int	Locale à main

Et dernierement pour MA5 :

Variable	Utilité	Type	Portée
tab	Tableau de chaînes de caractères pour stocker des lettres de l'alphabet	string array	Locale à main
tab1	Tableau de chaînes de caractères pour stocker des valeurs binaires correspondant à chaque lettre de l'alphabet	string array	Locale à main
entree	Variable pour stocker le numéro d'agent entré par l'utilisateur	int	Locale à main
id	Variable pour stocker l'ID de la carte insérée par l'utilisateur	string	Locale à main
t	Variable booléenne pour contrôler la boucle	bool	Locale à main

Voici le lien du flogorithme pour le tester si besoin :

Algorithme_livrable_
3.fprg

Pour conclure, Le programme principal nous permet ainsi de sélectionner et faire fonctionner le bon niveau de sécurité correspondant à l'agent entré et son module. Cela nous permettra identifier l'agent de manière sécurisée.