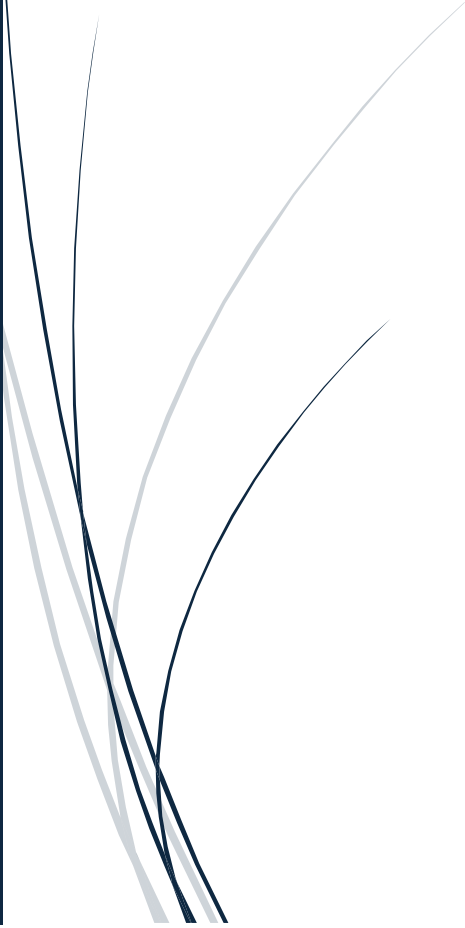


25/09/2025

Compte Rendu TD2



Maxime Brodin
3A - CYBER

Table des matières

I. Contexte	2
1. Objectifs	2
2. Consignes	2
3. Fonctions prédéfinies	2
II. Fonctions de base	2
1. EGAL	2
2. DANSLISTE	5
3. POGNON	7
III. Jeu de base	8
1. Menu	8
2. TOUR	11
IV. Extension	16
1. Timer	16
V. Annexes	18
1. Format de fichier	18

I. Contexte

1. Objectifs

L'objectif de cet exercice est d'écrire un algorithme de "menu" en console avec vérification de saisie.

Le support sera un jeu de "questions".

Le joueur a 10 tours de jeu. A chaque tour, une question est posée, et 4 réponses possibles sont données.

Si la réponse est valide, le joueur remporte le tour, et la cagnotte est augmentée.

Si la réponse n'est pas valide, le joueur perd l'ensemble de la cagnotte.

A chaque tour, le joueur peut décider de tenter la question ou d'abandonner et ainsi encaisser la cagnotte.

2. Consignes

Sont demandés, pour le programme principal et chaque fonction :

- l'algorithme, en langage "naturel"
- un algorithme

3. Fonctions prédéfinies

Les fonctions suivantes sont fournies :

- `VERIF` qui prend deux paramètres : une valeur et un type sous la forme d'un entier

- 0 : chaîne
- 1 : entier
- 2 : flottant

- `LSQUESTIONS` qui donne la liste des questions disponibles, sous forme de clefs

- `UNEQUESTION` qui, à partir d'une clef de question, renvoie une liste sous la forme : `[Contenu, Réponse A, Réponse B, Réponse C, Réponse D, Réponse valide]`

- `MAJ` qui transforme une chaîne en majuscules

- `MIN` qui transforme une chaîne en minuscules

- `MST` qui renvoie le timestamp UNIX actuel

- `LIRENB` qui effectue une lecture ****non-bloquante**** d'une saisie utilisateur. Cette fonction prend un paramètre : le nom d'une fonction à exécuter une fois qu'une saisie utilisateur est détectée et validée

II. Fonctions de base

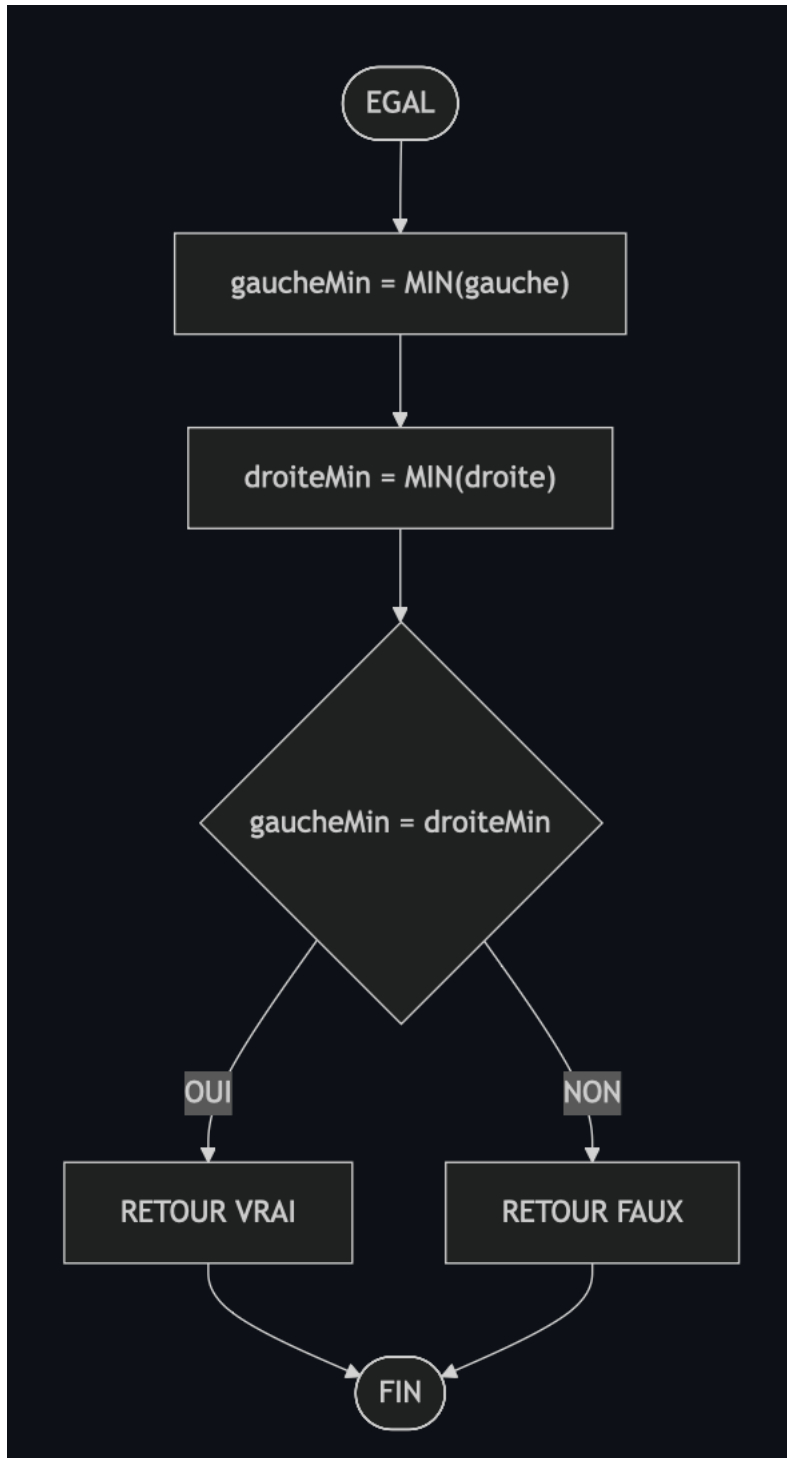
1. EGAL

Créez la fonction `EGAL` qui prend deux paramètres :

- `gauche`, une chaîne
- `droite`, une chaîne

Cette fonction vérifie si les deux chaînes sont égales en ignorant la casse.

ALGORIGRAMME :



ALGRORITHME :

```
DEBUT EGAL
  PARAM CHAINE gauche
  PARAM CHAINE droite
  VARIABLE CHAINE gaucheMin
  VARIABLE CHAINE droiteMin

  gaucheMin ← MIN(gauche)
  droiteMin ← MIN(droite)

  SI gaucheMin = droiteMin ALORS
    RETOUR VRAI
  SINON
    RETOUR FAUX
  FIN SI
FIN
```

J'ai choisi d'utiliser la fonction prédéfinie MIN() plutôt que MAJ() car convertir en minuscules est plus naturel. L'algorithme est assez direct : on convertit les deux chaînes, puis on compare.

La principale difficulté que j'ai rencontrée était de décider où faire la conversion. J'ai d'abord pensé à convertir directement dans la condition du SI, mais ça rendait le code moins lisible, donc j'ai pris des variables intermédiaires (gaucheMin et droiteMin) qui rendent l'algorithme plus clair et plus facile à débbug.

Pour l'algorigramme, j'ai voulu garder une structure simple et linéaire. La condition compare directement les deux chaînes converties, et les deux branches mènent vers des retours différents avant de converger vers la fin.

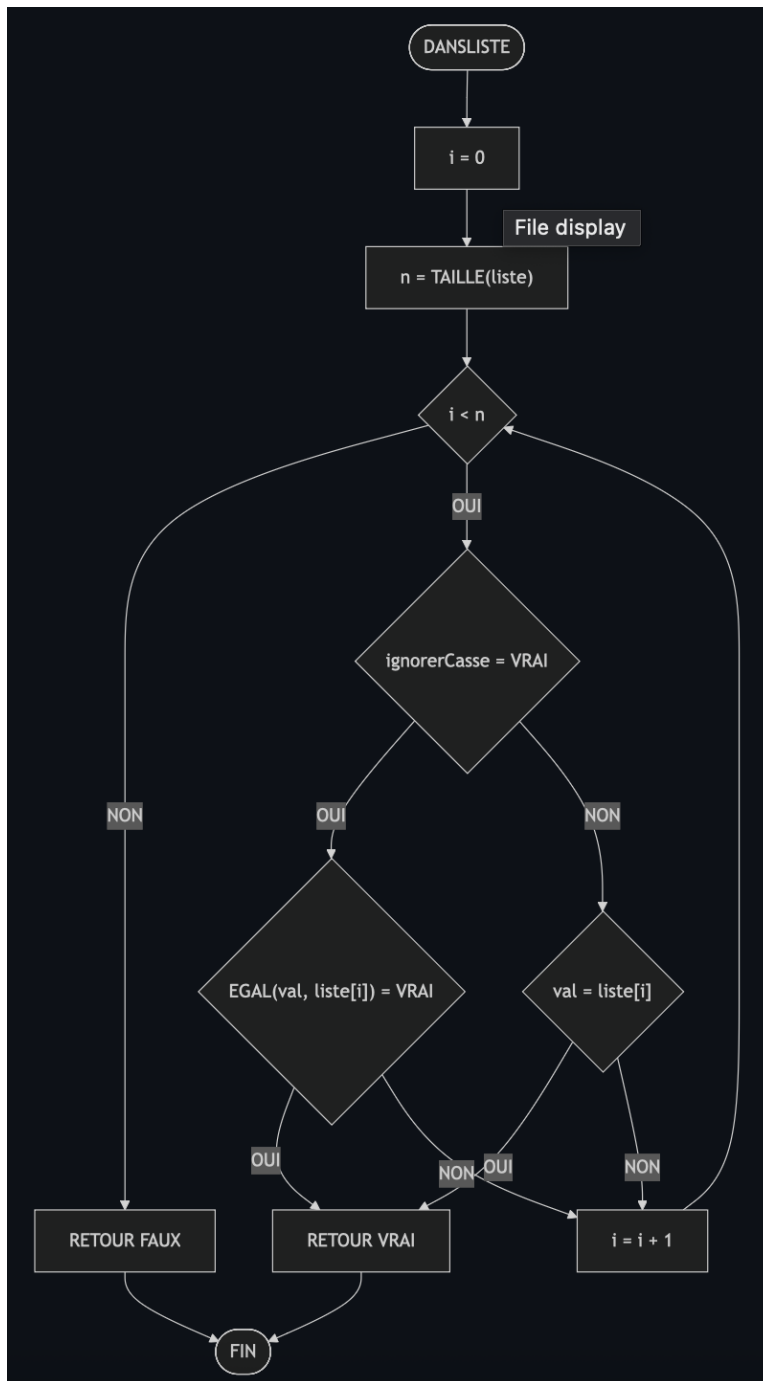
2. DANSLISTE

Créez la fonction `DANSLISTE` qui prend trois paramètres (dont un facultatif) :

- `val`, la valeur à vérifier
- `liste`, la liste
- `ignorerCasse`, un booléen pour ignorer la casse (`VRAI`) ou non (`FAUX`) - par défaut `FAUX`

Cette fonction vérifie si la valeur `val` est dans la liste `liste` et renvoie un booléen.

ALGORIGRAMME :



ALGORITHME :

```
DEBUT DANSLISTE
  PARAM val
  PARAM LISTE liste
  PARAM BOOLEEN ignorerCasse DEFAULT FAUX
  VARIABLE ENTIER i
  VARIABLE ENTIER n

  i ← 0
  n ← TAILLE(liste)

  TANT QUE i < n FAIRE
    SI ignorerCasse = VRAI ALORS
      SI EGAL(val, liste[i]) = VRAI ALORS
        RETOUR VRAI
      FIN SI
    SINON
      SI val = liste[i] ALORS
        RETOUR VRAI
      FIN SI
    FIN SI
    i ← i + 1
  FIN TANT QUE

  RETOUR FAUX
FIN
```

Ici, le truc qui m'a pris du temps, c'est de décider si je devais faire deux boucles séparées ou une seule avec un test à l'intérieur. Du coup j'ai pris une seule boucle avec un SI qui teste le mode de comparaison à chaque itération. C'est peut-être pas optimal niveau performance, mais ça rend le code plus compact.

Pour l'algorithme, j'ai galéré sur la représentation du double test. Il fallait montrer clairement que selon la valeur d'ignorerCasse, on utilise soit EGAL (pour ignorer la casse) soit une comparaison directe. J'ai fini par mettre deux losanges de décision qui convergent vers les mêmes actions.

La partie la plus délicate était de bien réutiliser la fonction EGAL qu'on venait de créer. Ça évite de dupliquer la logique de comparaison insensible à la casse, et ça montre qu'on peut construire des fonctions qui s'appuient sur d'autres.

J'ai stocké la taille de la liste dans une variable n pour éviter de la recalculer à chaque tour de boucle, même si techniquement `TAILLE()` devrait être assez rapide.

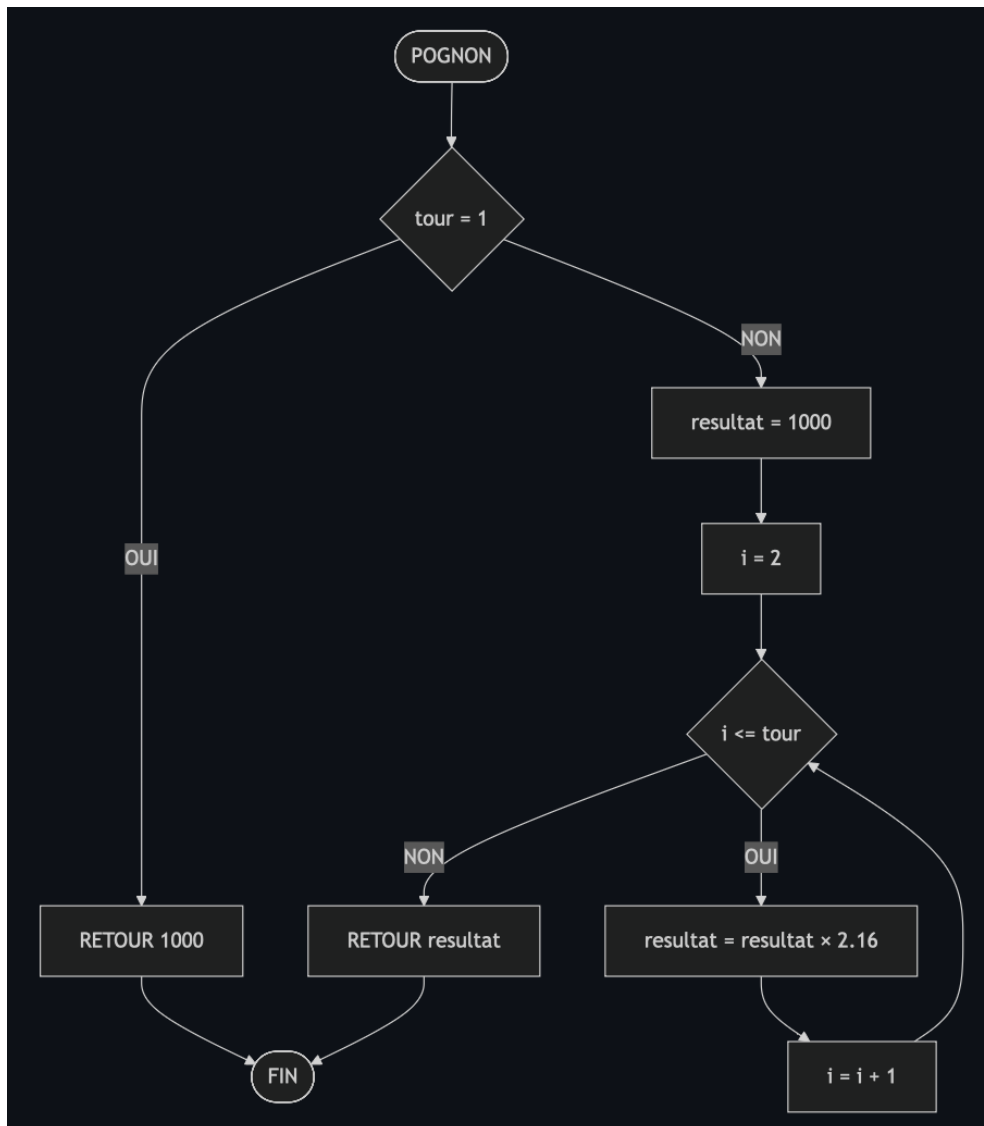
3. POGNON

Créez la fonction 'POGNON' qui prend un paramètre entier 'tour' et renvoie l'argent espéré pour le tour, selon la formule mathématique :

$$a(n) = a(n-1) \cdot 2.16$$

Avec n étant le numéro du tour, sachant que $a(1) = 1000$.

ALGORIGRAMME :



ALGRORITHME :

```
DEBUT POGNON
  PARAM ENTIER tour
  VARIABLE REEL resultat
  VARIABLE ENTIER i

  SI tour = 1 ALORS
    RETOUR 1000
  SINON
    resultat ← 1000
    POUR i DE 2 A tour FAIRE
      resultat ← resultat × 2.16
    FIN POUR
    RETOUR resultat
  FIN SI
FIN
```

POGNON calcule la cagnotte selon une formule exponentielle. J'ai commencé par identifier le cas de base : tour 1 = 1000€. Pour les autres tours, il faut appliquer la multiplication par 2.16 de façon répétée.

Au début j'ai commencé par faire une boucle simple, mais il fallait optimiser le cas tour = 1 pour éviter des calculs inutiles donc j'ai ajouté une condition qui retourne directement 1000 pour le premier tour.

Pour l'algorithme, j'ai hésité entre une boucle POUR et TANT QUE. J'ai choisi POUR parce que le nombre d'itérations est connu à l'avance.

Le plus important était de bien initialiser les variables. La variable résultat démarre à 1000 et on commence la boucle au tour 2. Ça évite de faire une multiplication de trop et ça simplifie la logique.

III. Jeu de base

1. Menu

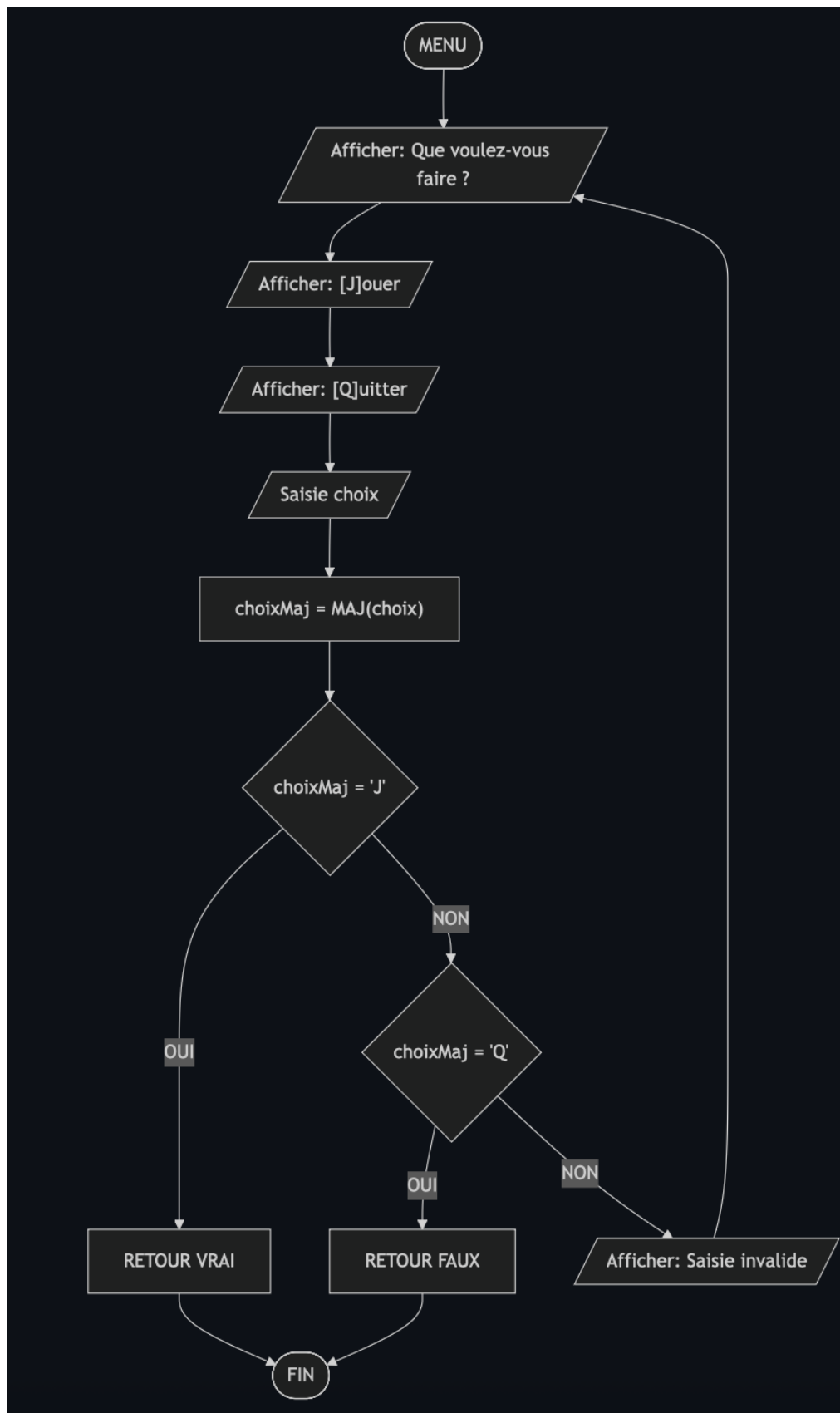
Créez le menu initial, afin de demander à l'utilisateur s'il souhaite :

- `[J]ouer`
- `[Q]uitter`

Si l'utilisateur choisit "Q" (peu importe la casse), le programme s'arrête.
Si l'utilisateur choisit "J", le jeu se lance au premier tour.

Si la saisie ne correspond à rien, demandez à nouveau à l'utilisateur.

ALGORIGRAMME :



ALGRORITHME :

```
DEBUT MENU
  VARIABLE CHAINE choix
  VARIABLE CHAINE choixMaj
  VARIABLE BOOLEEN continuer ← VRAI

  TANT QUE continuer = VRAI FAIRE
    ECRIRE "Que voulez-vous faire ?"
    ECRIRE "[J]ouer"
    ECRIRE "[Q]uitter"
    LIRE choix

    choixMaj ← MAJ(choix)

    SI choixMaj = "J" ALORS
      RETOUR VRAI
    SINON SI choixMaj = "Q" ALORS
      RETOUR FAUX
    SINON
      ECRIRE "Saisie invalide, veuillez recommencer."
    FIN SI
  FIN TANT QUE
FIN
```

Ici j'ai pris TANT QUE qui tourne jusqu'à ce qu'on ait une saisie valide.

Le truc qui m'a posé problème au début c'est la variable "continuer". Je me suis dit que c'était peut-être pas nécessaire vu qu'on fait des RETOUR dans la boucle, mais au final ça rend l'algorithme plus clair à lire parce que sans cette variable, on aurait eu une boucle infinie avec juste des RETOUR dedans.

J'ai utilisé MAJ() pour normaliser la saisie, comme ça peu importe si l'utilisateur tape "j", "J" ou même "jouer". Le choix entre VRAI pour jouer et FAUX pour quitter était logique : VRAI = on continue le programme, FAUX = on s'arrête.

Pour l'algorithme, j'ai voulu montrer la boucle qui revient au début en cas de saisie invalide. La structure de conditions successives permet de bien voir les deux tests (J puis Q) avant de revenir à l'affichage du menu si rien ne correspond.

2. TOUR

Créez la fonction `TOUR`, qui prend un paramètre entier, le numéro du tour, ainsi que la liste des questions déjà posées (sous forme de clefs).

Dans cette fonction, obtenez une question qui n'a pas déjà été posée.

Puis, affichez la question, suivie des réponses, et attendez la saisie de l'utilisateur.

Une fois la saisie effectuée, si celle-ci correspond à une des réponses (A, B, C ou D), affichez le résultat.

Si le joueur a perdu, revenez au menu principal.

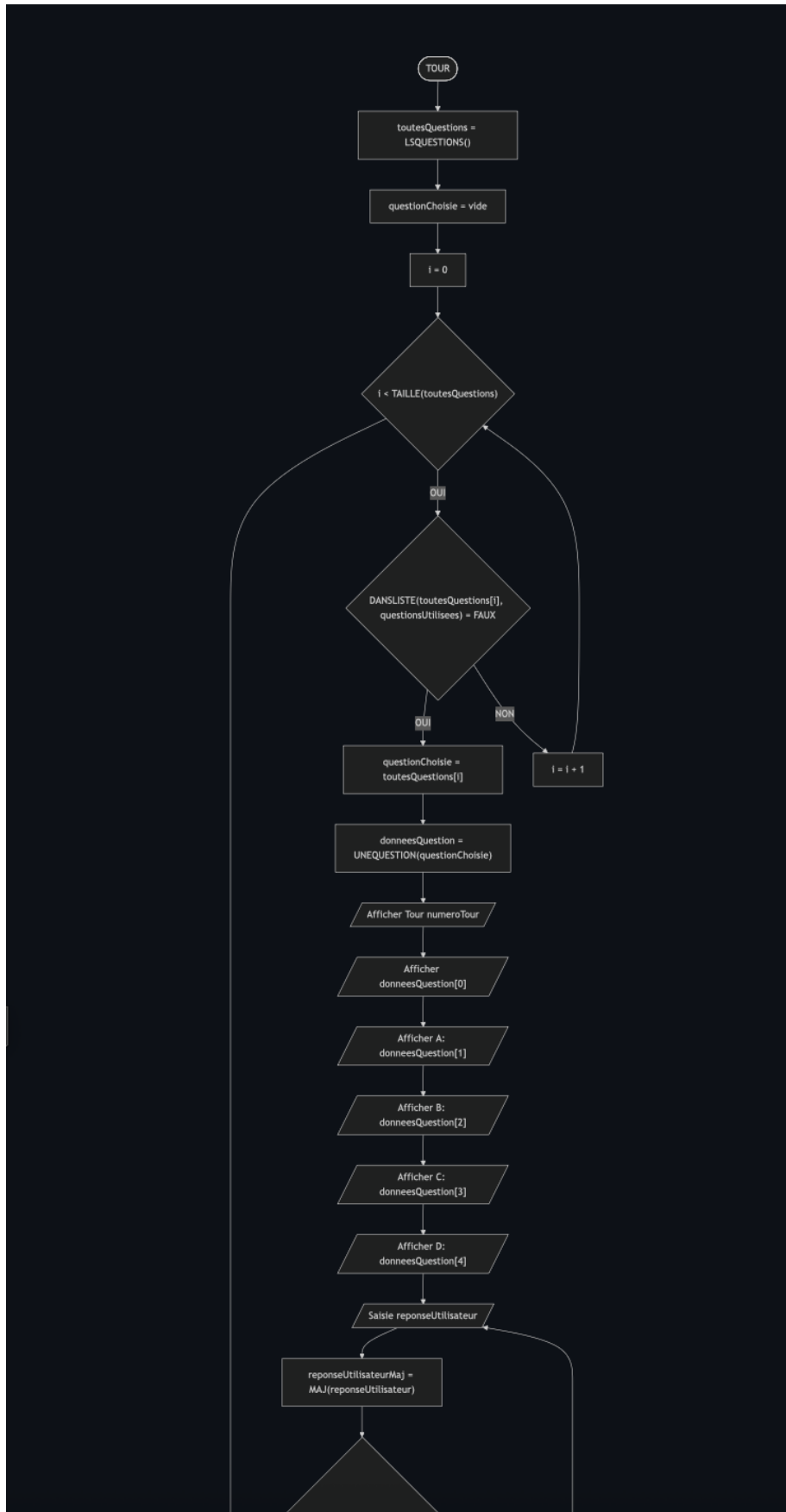
Si le joueur a gagné, affichez la cagnotte, puis (si le tour est inférieur à 10) demandez si le joueur souhaite `[c]ontinuer` ou `[a]rrêter`.

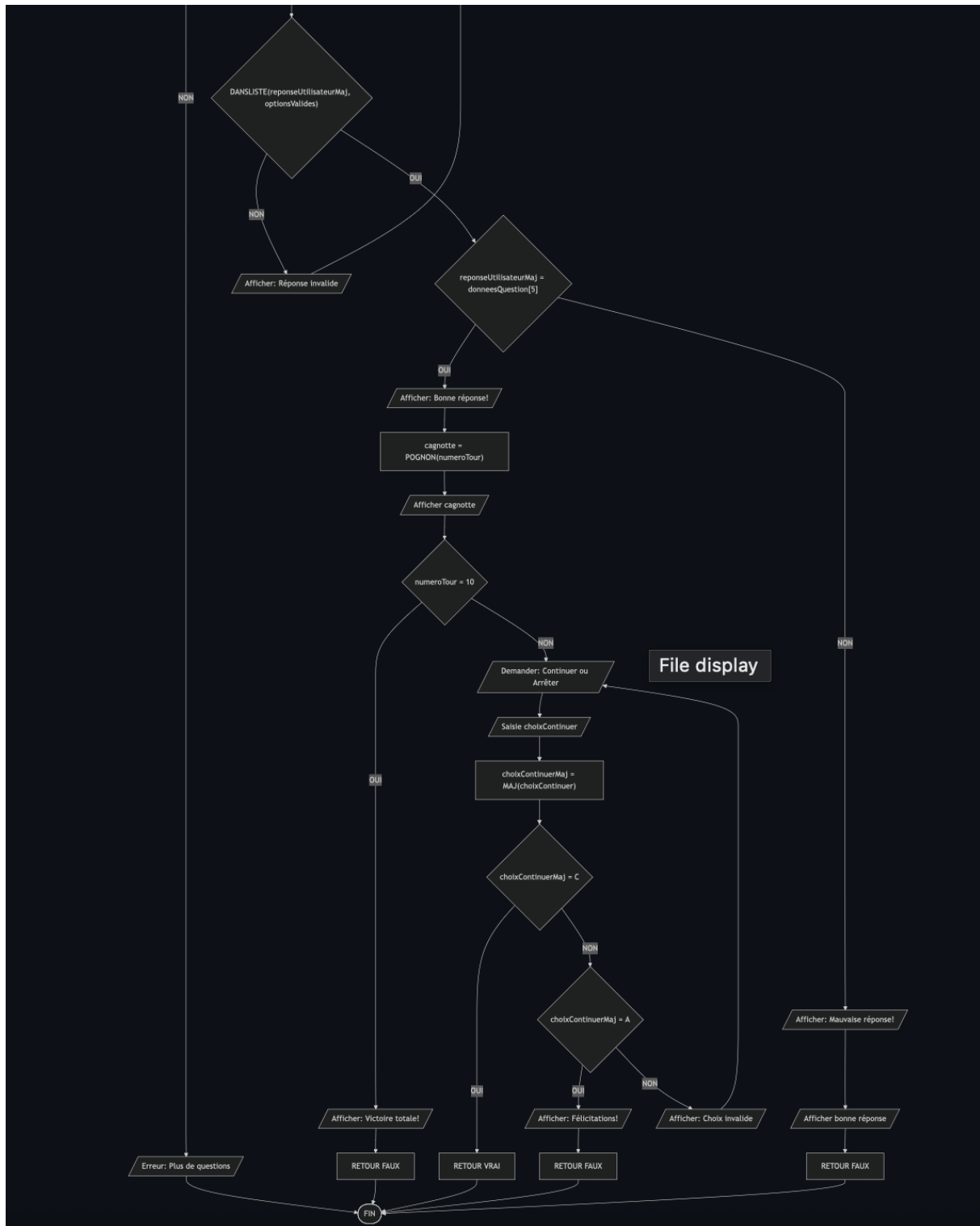
Si le joueur arrête, il remporte la cagnotte et le programme s'arrête.

Si le joueur continue, passez au tour suivant.

Si le joueur gagne au dernier tour, le programme s'arrête.

ALGORIGRAMME :





ALGORITHME :

```
DEBUT TOUR
  PARAM ENTIER numeroTour
  PARAM LISTE questionsUtilisees
  VARIABLE LISTE toutesQuestions
  VARIABLE CHAINE questionChoisie
  VARIABLE ENTIER i
  VARIABLE LISTE donneesQuestion
  VARIABLE CHAINE reponseUtilisateur
  VARIABLE CHAINE reponseUtilisateurMaj
  VARIABLE LISTE optionsValides
  VARIABLE REEL cagnotte
  VARIABLE CHAINE choixContinuer
  VARIABLE CHAINE choixContinuerMaj
  VARIABLE BOOLEEN continuer

  toutesQuestions ← LSQUESTIONS()
  questionChoisie ← ""
  optionsValides ← ["A", "B", "C", "D"]

  POUR i DE 0 A TAILLE(toutesQuestions) - 1 FAIRE
    SI DANSLISTE(toutesQuestions[i], questionsUtilisees) = FAUX ALORS
      questionChoisie ← toutesQuestions[i]
      SORTIR DE BOUCLE
    FIN SI
  FIN POUR
```

```
SI questionChoisie = "" ALORS
  ECRIRE "Erreur: Aucune question disponible"
  RETOUR FAUX
FIN SI

donneesQuestion ← UNEQUESTION(questionChoisie)

ECRIRE "=== TOUR ", numeroTour, " ==="
ECRIRE donneesQuestion[0]
ECRIRE "A: ", donneesQuestion[1]
ECRIRE "B: ", donneesQuestion[2]
ECRIRE "C: ", donneesQuestion[3]
ECRIRE "D: ", donneesQuestion[4]

continuer ← VRAI
TANT QUE continuer = VRAI FAIRE
  ECRIRE "Votre réponse (A, B, C ou D): "
  LIRE reponseUtilisateur
  reponseUtilisateurMaj ← MAJ(reponseUtilisateur)

  SI DANSLISTE(reponseUtilisateurMaj, optionsValides) = VRAI ALORS
    continuer ← FAUX
  SINON
    ECRIRE "Réponse invalide! Veuillez saisir A, B, C ou D."
  FIN SI
FIN TANT QUE

SI reponseUtilisateurMaj = donneesQuestion[5] ALORS
  ECRIRE "Bonne réponse!"
  cagnotte ← POGNON(numeroTour)
  ECRIRE "Cagnotte actuelle: ", cagnotte, " €"
```

```
SI questionChoisie = "" ALORS
  ECRIRE "Erreur: Aucune question disponible"
  RETOUR FAUX
FIN SI

donneesQuestion ← UNEQUESTION(questionChoisie)

ECRIRE "=== TOUR ", numeroTour, " ==="
ECRIRE donneesQuestion[0]
ECRIRE "A: ", donneesQuestion[1]
ECRIRE "B: ", donneesQuestion[2]
ECRIRE "C: ", donneesQuestion[3]
ECRIRE "D: ", donneesQuestion[4]

continuer ← VRAI
TANT QUE continuer = VRAI FAIRE
  ECRIRE "Votre réponse (A, B, C ou D): "
  LIRE reponseUtilisateur
  reponseUtilisateurMaj ← MAJ(reponseUtilisateur)

  SI DANSLISTE(reponseUtilisateurMaj, optionsValides) = VRAI ALORS
    continuer ← FAUX
  SINON
    ECRIRE "Réponse invalide! Veuillez saisir A, B, C ou D."
  FIN SI
FIN TANT QUE

SI reponseUtilisateurMaj = donneesQuestion[5] ALORS
  ECRIRE "Bonne réponse!"
  cagnotte ← POGNON(numeroTour)
  ECRIRE "Cagnotte actuelle: ", cagnotte, " €"
```

Alors là c'était plus difficile. C'est clairement la fonction la plus compliquée du projet avec tous les cas à gérer.

D'abord le système de sélection de question. J'ai récupéré toutes les questions puis cherché la première qui n'était pas dans la liste des utilisées. Pas très optimisé mais ça marche et c'est lisible.

Ensuite la validation des réponses. J'ai créé une liste des options valides pour réutiliser DANSLISTE, ça évite de refaire la validation à la main. La boucle TANT QUE force l'utilisateur à donner une réponse correcte.

Le plus galère c'était la gestion des fins de partie : mauvaise réponse (direct au menu), bonne réponse au tour 10 (victoire), ou bonne réponse avant le tour 10 (choix continuer/arrêter). J'ai dû bien réfléchir aux retours pour que le programme principal comprenne quoi faire ensuite.

Pour l'algorithme, j'ai essayé de garder une logique séquentielle même si c'est devenu assez énorme avec tous les embranchements. Les boucles de validation créent des cycles dans le diagramme mais ça reste quand même compréhensible.

IV.Extension

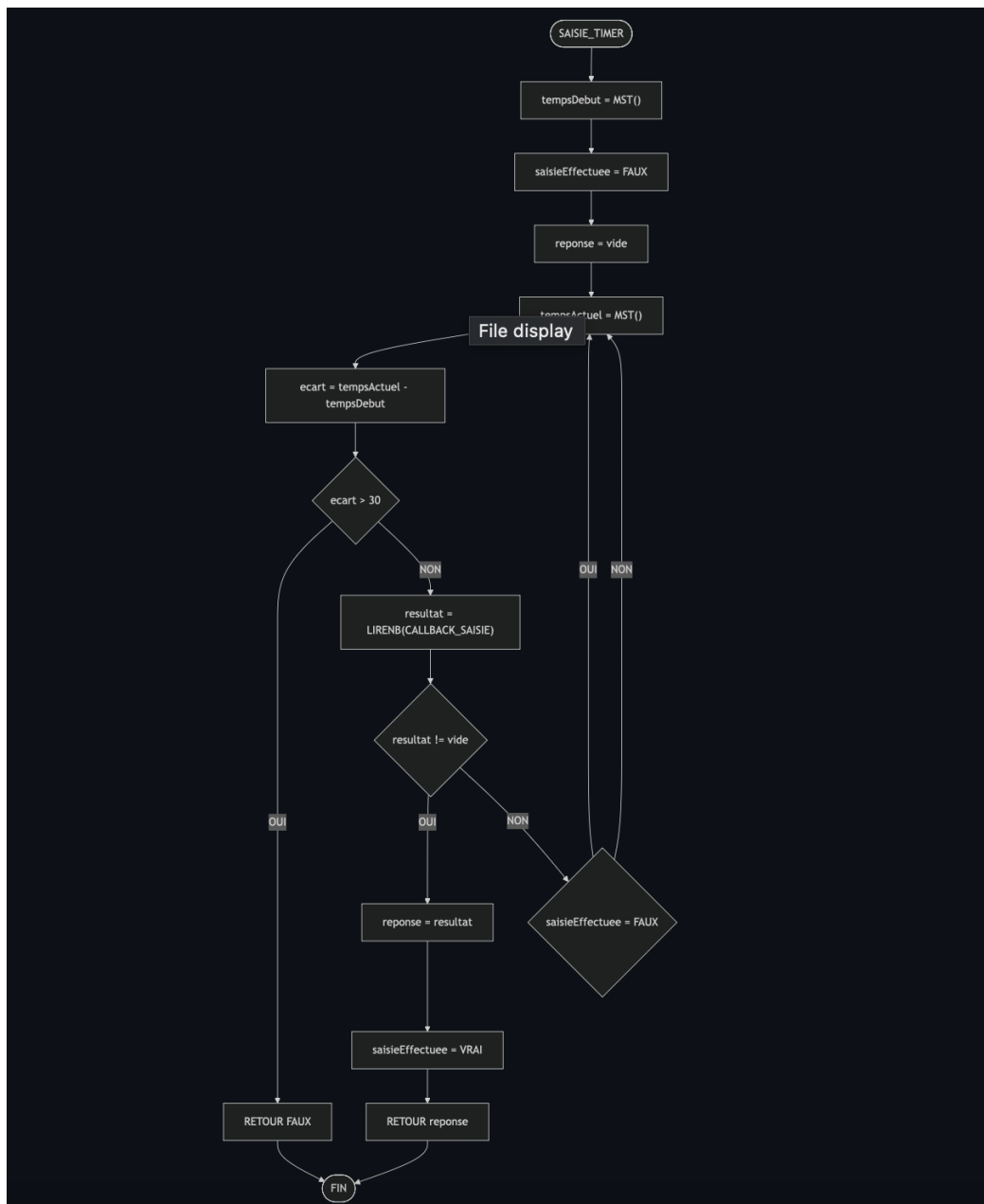
1. Timer

Implémentez le timer.

Pour cela, utilisez la fonction `LIRENB` pour effectuer une lecture non-bloquante, encapsulée dans une boucle `TANTQUE` vérifiant un écart de temps.

Si cet écart de temps est supérieur à 30, et qu'aucune saisie n'a été réalisée, la réponse est considérée comme fausse.

ALGORIGRAMME :



ALGORITHMME :

```
DEBUT SAISIE_TIMER
  VARIABLE ENTIER tempsDebut
  VARIABLE ENTIER tempsActuel
  VARIABLE ENTIER ecart
  VARIABLE BOOLEEN saisieEffectuee
  VARIABLE CHAINE reponse
  VARIABLE CHAINE resultat

  tempsDebut ← MST()
  saisieEffectuee ← FAUX
  reponse ← ""

  TANT QUE saisieEffectuee = FAUX FAIRE
    tempsActuel ← MST()
    ecart ← tempsActuel - tempsDebut

    SI ecart > 30 ALORS
      RETOUR FAUX
    FIN SI

    resultat ← LIRENB(CALLBACK_SAISIE)
    SI resultat != "" ALORS
      reponse ← resultat
      saisieEffectuee ← VRAI
    FIN SI
  FIN TANT QUE

  RETOUR reponse
FIN
```

```
DEBUT CALLBACK_SAISIE
  VARIABLE CHAINE saisie
  LIRE saisie
  RETOUR saisie
FIN
```

Ici, l'extension, c'était pas simple. La lecture non-bloquante avec LIRENB, j'ai vraiment galéré à comprendre comment ça fonctionnait.

A la base, je voulais boucler en vérifiant le temps et en essayant de lire. Mais implémenter ça proprement, c'est une autre histoire.

J'ai dû créer une fonction callback séparée parce que LIRENB en a besoin.

C'était un peu compliqué de gérer les états : savoir si une saisie a été faite ou pas, calculer l'écart de temps correctement. J'ai opté pour un flag booléen `saisieEffectuee` pour contrôler la boucle.

Je sais que l'algorithme est grand avec toutes ces conditions, mais au final ça reflète bien la logique : on vérifie le temps, on essaie de lire, on recommence jusqu'à avoir une réponse ou dépasser 30 secondes.

V. Annexes

1. Format de fichier

Note

Cet encart est donné à titre informatif

Chaque question est dans un format dédié, dans un fichier .qdpd. Le nom du fichier est utilisé en tant que clef (le fichier abc.qdpd a donc pour clef abc).

===

Contenu de la question

=A=

Réponse A

=B=

Réponse B

=C=

Réponse C

=D=

Réponse D

=X=

X est remplacé par A, B, C ou D selon la bonne réponse.