

Sara CRUBELLIER
Yannick DUPUIS
Camille PAPILLIER

Institut Charles Cros, Université de Marne-la-Vallée
Avec la collaboration de l'institut Dysphasia

SAMASIA

ANNEXES

Projet tutoré

Formation d'Ingénieur IMAC

Master Arts & Sciences de l'Enregistrement

Année scolaire 2004/2005

SOMMAIRE

GESTION DES PROFILS	3
– Structure du XML	
– Gestion du fichiers	
– Gestion du XML	
– Gestion des profils	
– Chargement d'un profil	
– Création d'un profil	
JEU DES PHRASES A RECONSTITUER	22
– Extraits du fichier jeu1.xml	
– Gestion des phrases	
– Initialisation du jeu1.xml	
– Création des objets constituant les mots	
JEU DE LA CORRECTION SYNTAXIQUE	35
– Version partielle du fichier XML jeu2.xml	
– Initialisation des variables pour le niveau 1	
– Parsage du fichier XML, sélection aléatoire puis affichage d'une phrase	
– Bouton de validation après complétion du champ de texte éditable	
– Vérification de la réponse puis traitement en fonction du résultat	
– Réinitialisation des variables et contrôle de l'avancement du jeu	
– Calcul du score	
– Bouton d'affichage de l'aide	
JEU DE DISCRIMINATION DES PHONEMES AUDITIVEMENT PROCHES	39
– Un fichier XML codant le niveau 2	
– Chargement du XML et sélection du nœud de catégorie X	
– Exemple du code d'un bouton de choix de possibilités	
– Code du bouton qui lance le son	
– Code lorsque le joueur n'a pas réussi	
– Code du script lorsque le joueur a réussi	

GESTION DES PROFILS

Structure du XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<profils>
  <joueur id="test">

    <niveau id="1" score="31031980">
      <jeu id="11" statut="1" nbParties="31" />
      <jeu id="12" statut="1" nbParties="3" />
      <jeu id="13" statut="1" nbParties="1980" />
    </niveau>

    <niveau id="2" score="400">
      <jeu id="21" statut="1" nbParties="31" />
      <jeu id="22" statut="0" nbParties="03" />
      <jeu id="23" statut="0" nbParties="1980" />
    </niveau>
  </joueur>
</profils>
```

Gestion du fichier

- Fonction renvoyant le contenu du fichier sous rme d'une chaîne de caractère

```
on ReadFile( sFilePath )

-----
-- Création de l'instance de l'Xtra fileIO --
-----
oFile = new xtra( "fileio" )

-----
-- Ouverture du fichier --
-----
oFile.openFile( sFilePath, 1 )

-- Cas où le chemin est invalide
if oFile.status() then

  put "ERREUR (ReadFile): chemin du fichier invalide"
  return #cheminInvalide

end if

-----
-- Lecture du fichier --
-----
sContents = oFile.readFile()

-- Cas où le fichier ne peut pas être lu
if oFile.status() then

  put "ERREUR (ReadFile): fichier illisible"
  sContents = #contenuIllisible

end if

-----
```

```

-- Fermeture du fichier --
-----
oFile.closeFile()

-- On renvoie la chaine de caractère qui correspond au contenu du fichier
return sContents

end ReadFile

```

```

-----
-- Fonction enregistrant le contenu d'une chaine de caractère dans un fichier
spécifié --
-----

```

```

on SaveFile( sFilePath, sContents )

```

```

-----
-- Création de l'instance de l'Xtra fileIO --
-----

```

```

oFile = new xtra( "fileio" )

```

```

-----
-- Destruction de l'ancien fichier --
-----

```

```

oFile.openFile( sFilePath, 0 )

```

```

oFile.delete()

```

```

-- Ouverture et écriture dans le fichier

```

```

oFile.createFile( sFilePath )

```

```

oFile.openFile( sFilePath, 0 )

```

```

oFile.writeString( sContents )

```

```

-----
-- Fermeture du fichier --
-----

```

```

oFile.closeFile()

```

```

return 1

```

```

end SaveFile

```

Gestion du XML

```

-----
-- Fonction générant une liste de propriétés à partir d'une chaine de caractères
au format XML --
-----

```

```

on ParseXml( sXml )

```

```

-----
-- Création de l'instance de l'Xtra xmlParser --
-----

```

```

oXml = new xtra( "xmlparser" )

```

```

oXml.parseString( sXml )

```

```

lpXml = oXml.makePropList() -- Création de la liste de propriété
return lpXml -- On renvoie la liste de propriété correspondant au contenu du
fichier XML
end ParseXml

```

```

-----
-- Fonction convertissant une liste de propriétés en chaîne de caractères avec
un contenu en XML --
-----

```

```

on makeStringXml( lpXml, sXml, iNumTab )

iChildNumber = count( lpXml.child )

```

```

-----
-- Cas où l'on est à la fin d'une branche (pivot de fin de la récursivité) --
-----

```

```

if iChildNumber = 0 then

-- Ajout des tabulations en fonction du niveau de profondeur de l'arbre
récursif
repeat with n = 0 to iNumTab

sXml = sXml &TAB

end repeat

-- Ouverture de la balise
sXml = sXml &"<" &lpXml.name

iAttNumber = count( lpXml.attributes )

-- On parcourt tous les attributs
repeat with n = 1 to iAttNumber

sXml = sXml &" " &lpXml.attributes.getpropat( n ) &"=" &QUOTE
&lpXml.attributes[n] &QUOTE

end repeat

-- fermeture de la balise
return sXml &">" &RETURN

end if

```

```

-----
-- Cas où l'on est sur un nouveau noeux --
-----

```

```

-- Ajout des tabulations en fonction du niveau de profondeur de l'arbre
récursif
repeat with n = 0 to iNumTab

sXml = sXml &TAB

end repeat

-- Ouverture de la balise
sXml = sXml &"<" &lpXml.name

iAttNumber = count( lpXml.attributes )

-- On parcourt tous les attributs
repeat with n = 1 to iAttNumber

```

```

    sXml = sXml &" " &lpXml.attributes.getpropat( n ) &"=" &QUOTE &
lpXml.attributes[n] &QUOTE

end repeat

sXml = sXml &">" &RETURN

-----
-- Traitement de tous les enfants du noeux --
-----
repeat with n = 1 to iChildNumber

    sXml = MakeStringXml( lpXml.child[n], sXml, iNumtab+1 )

end repeat

-- Ajout des tabulations en fonction du niveau de profondeur de l'arbre
récursif
repeat with n = 0 to iNumTab

    sXml = sXml &TAB

end repeat

-- Fermeture de la balise du noeud
sXml = sXml &"</" &lpXml.name &">" &RETURN

return sXml

end MakeStringXml

```

Gestion des profils

```

-----
-- Fonction chargeant sous la forme d'une liste de propriétés les paramètres du
joueur spécifié en argument --
-----

on LoadProfile( sProfileId )

    -----
    -- Initialisation des paramètres --
    -----

    sFilePath = "profils.xml"

    -----
    -- Chargement de la liste des profils --
    -----

    sXml = ReadFile( sFilePath ) -- Lecture du fichier XML contenant les profils
    lpXml= ParseXml( sXml ) -- Conversion du contenu du fichier XML en liste de
propriétés

    -----
    -- Vérification de la structure des listes de propriétés --
    -----

    -- Cas où la liste de propriétés des profils est invalide
    if IsProfileList( lpXml ) = 0 then

```

```

    put "ERREUR (LoadProfile): liste de propriétés des profils invalide"
    return 0
end if

iNbPlayer = count( lpXml.child )

-----
-- Recherche du profil --
-----
repeat with n = 1 to iNbPlayer

    -- Cas où le profil existe déjà
    if lpXml.child[n].attributes.id = sProfileId then

        put "Chargement du profil de: " &sProfileId

        return lpXml.child[n] -- On renvoie la partie de la liste correspondant au
joueur recherché

    end if

end repeat

-- Cas où le profil n'existe pas
put "ERREUR (LoadProfile): impossible de charger le profil: " &sProfileId

return 0
end LoadProfile

-----
-- Fonction ajoutant un nouveau profil --
-----

on CreateProfile( sPlayerName )

-----
-- Initialisation des paramètres --
-----

sFilePath = "profils.xml"
iNbLevel = 2
iNbGame = 3

-----
-- Chargement de la liste des profils --
-----

sXml = ReadFile( sFilePath ) -- Lecture du fichier XML contenant les profils
lpXml= ParseXml( sXml ) -- Conversion du contenu du fichier XML en liste de
propriétés

-----
-- Vérification de la structure des listes de propriétés --
-----

-- Cas où la liste de propriétés des profils est invalide
if IsProfileList( lpXml ) = 0 then

    put "ERREUR (CreateProfile): liste de propriétés des profils invalide"
    return 0
end if

```

```

end if

-----
-- Création du profil vierge avec l'identifiant spécifié --
-----

-- Création de la balise joueur
lpPlayerAttributes = [#id:sPlayerName]
lpPlayer = [#name:"joueur", #attributes:lpPlayerAttributes, #child:[]]

-- Création des balises pour chaque niveau de jeu (2 par défaut)
repeat with l = 1 to iNbLevel

    lpLevelAttributes = [#id:l, #score:"0"]
    lpLevel = [#name:"niveau", #attributes:lpLevelAttributes, #child:[]]

    -- Création des balises pour les jeux (3 par défaut pour chaque niveau)
    repeat with g = 1 to iNbGame

        lpGameAttributes = [#id:g+l*10, #statut:"0", #nbParties:"0"]
        lpGame = [#name:"jeu", #attributes:lpGameAttributes, #child:[]]

        -- ajout du Jeu au niveau
        lpLevel.child.add( lpGame )

    end repeat

    -- Ajout du niveau au joueur
    lpPlayer.child.add( lpLevel )

end repeat

-----
-- Insertion du nouveau joueur dans la liste de propriétés des profils --
-----

lpXml.child.add( lpPlayer )

-----
-- Conversion de la liste de propriétés des profils en chaîne de caractères
-----

sXml = "<?xml version=" &QUOTE &"1.0" &QUOTE &" encoding=" &QUOTE &"ISO-8859-
1" &QUOTE &" ?>" &RETURN &RETURN &MakeStringXml(lpXml, "", 0)

if SaveFile( sFilePath, sXml ) = 0 then

    put "ERREUR (CreateProfile): Impossible d'enregistrer le nouveau profil dans
le fichier XML des profils"
    return 0

end if

-- Cas où la mise à jour s'est correctement effectuée
put "Nouveau profil créé"

-----
-- Préparation des données de retour de la fonction --
-----

iNbPlayer = count( lpXml.child ) -- On retrouve la position du profil créé
dans la liste

return lpXml.child[iNbPlayer] -- On renvoie la liste de propriétés
correspondante au profil qui vient d'être créé

```

```
end CreateProfile
```

```
-----  
-- Fonction mettant à jour le profil principal à partir du profil d'un joueur  
-----
```

```
on SaveProfile( lpPlayer )
```

```
-----  
-- Initialisation des paramètres --  
-----
```

```
sFilePath = "profils.xml"
```

```
-----  
-- Chargement de la liste des profils --  
-----
```

```
sXml = ReadFile( sFilePath ) -- Lecture du fichier XML contenant les profils
```

```
lpXml = ParseXml( sXml ) -- Conversion du contenu du fichier XML en liste de  
propriétés
```

```
-----  
-- Vérification de la structure des listes de propriétés --  
-----
```

```
-- Cas où la liste de propriétés des profils est invalide
```

```
if IsProfileList( lpXml ) = 0 then
```

```
    put "ERREUR (SaveProfile): liste de propriétés des profils nvalide"  
    return 0
```

```
end if
```

```
-- Cas où la liste de propriétés du profil du joueur est invalide
```

```
if IsPlayerList( lpPlayer ) = 0 then
```

```
    put "ERREUR (SaveProfile): liste de propriétés du profil du joueur invalide"  
    return 0
```

```
end if
```

```
-----  
-- Recherche du profil du joueur --  
-----
```

```
iNbPlayer = count( lpXml.child )
```

```
-- Parcours de tous les joueurs
```

```
repeat with n = 1 to iNbPlayer
```

```
    lpOldPlayer = lpXml.child[n]
```

```
    -- Cas où le profil du joueur a été trouvé
```

```
    if lpOldPlayer.attributes.id = lpPlayer.attributes.id then
```

```
        -----  
        -- Mise à jour du profil du joueur --  
        -----
```

```
        iNbLevel = count( lpOldPlayer.child )
```

```

-- Parcours de tous les niveaux du joueur
repeat with l=1 to iNbLevel

    -- Mise à jour du score pour chaque niveau
    if UpdateScore( lpOldPlayer, l, lpPlayer.child[l].attributes.score ) = 0
then

        put "ERREUR (SaveProfile): Impossible de mettre à jour le niveau" &l
        return 0

    end if

    iNbGame = count( lpOldPlayer.child[l].child )

    -- Parcours de tous les jeux d'un niveau
    repeat with g=1 to iNbGame

        -- Mise à jour du statut pour chaque jeu
        if UpdateStatus( lpOldPlayer, g+iNbLevel*10, lpPlayer.child[l].child
[g].attributes.statut ) = 0 then

            put "ERREUR (SaveProfile): Impossible de mettre à jour le jeu: "
&g+iNbLevel*10
            return 0

        end if

        -- Mise à jour du nombre de parties pour chaque jeu
        if UpdateNbParties( lpOldPlayer, g+iNbLevel*10, lpPlayer.child[l].
child[g].attributes.nbParties ) = 0 then

            put "ERREUR (SaveProfile): Impossible de mettre à jour le jeu: "
&g+iNbLevel*10
            return 0

        end if

    end repeat

end repeat

-----
-- Conversion de la liste de propriétés des profils en chaîne de
caractères --
-----

sXml = "<?xml version=" &QUOTE &"1.0" &QUOTE &" encoding=" &QUOTE &"ISO-
8859-1" &QUOTE &" ?>" &RETURN &RETURN &MakeStringXml(lpXml, "", 0)

if SaveFile( sFilePath, sXml ) = 0 then

    put "ERREUR (SaveProfile): Impossible d'enregistrer le fichier"
    return 0

end if

-- Cas où la mise à jour s'est correctement effectuée

put "Enregistrement du fichier " &sFilePath
return 1

end if

end repeat

```

```

put "ERREUR (SaveProfile): profil du joueur introuvable"
return 0
end SaveProfile

-----
-- Fonction vérifiant la structure de la liste de propriétés au niveau du jeu
-----

on IsGameList( lpGame )

-- Cas où l'argument spécifié n'est pas une liste de propriétés
if ilk(lpGame) <> #proplist then

    put "ERREUR (IsGameList): l'argument n'est pas une liste de propriétés"
    return 0

end if

-- Cas où la structure de la liste de propriétés est incorrecte
if lpGame.getaprop( #name ) = void or lpGame.getaprop( #attributes ) = void or
lpGame.getaprop( #child ) = void then

    put "ERREUR (IsGameList):la liste de propriétés n'est pas correctement
structurée"
    return 0

end if

-- Cas où il n'y a pas les bons attributs dans la liste de propriétés
if lpGame.attributes.getaprop( #id ) = void or lpGame.attributes.getaprop
( #statut ) = void or lpGame.attributes.getaprop( #nbParties ) = void then

    put "ERREUR (IsGameList):la liste de propriétés n'a pas les bons attributs"
    return 0

end if

return 1
end IsGameList

-----
-- Fonction vérifiant la structure de la liste de propriétés au niveau du niveau
de jeu --
-----

on IsLevelList( lpLevel )

-- Cas où l'argument spécifié n'est pas une liste de propriétés
if ilk(lpLevel) <> #proplist then

    put "ERREUR (IsLevelList): l'argument n'est pas une liste de propriétés"
    return 0

end if

-- Cas où la structure de la liste de propriétés est incorrecte
if lpLevel.getaprop( #name ) = void or lpLevel.getaprop( #attributes ) = void
or lpLevel.getaprop( #child ) = void then

```

```

    put "ERREUR (IsLevelList):la liste de propriétés n'est pas correctement
structurée"
    return 0

end if

-- Cas où il n'y a pas les bons attributs dans la liste de propriétés
if lpLevel.attributes.getaprop( #id ) = void or lpLevel.attributes.getaprop
( #score ) = void then

    put "ERREUR (IsLevelList):la liste de propriétés n'a pas les bons attributs"
    return 0

end if

iNbGame = count( lpLevel.child )

-- Parcours de tous les jeux du niveau
repeat with n = 1 to iNbGame

    -- Cas où la structure de la liste n'est pas valide
    if IsGameList( lpLevel.child[n] ) = 0 then

        return 0

    end if

end repeat

return 1
end IsLevelList

```

```

-----
-- Fonction vérifiant la structure de la liste de propriétés au niveau du joueur
--
-----

```

```

on IsPlayerList( lpPlayer )

-- Cas où l'argument spécifié n'est pas une liste de propriétés
if ilk(lpPlayer) <> #proplist then

    put "ERREUR (IsPlayerList): l'argument n'est pas une liste de propriétés"
    return 0

end if

-- Cas où la structure de la liste de propriétés est incorrecte
if lpPlayer.getaprop( #name ) = void or lpPlayer.getaprop( #attributes ) =
void or lpPlayer.getaprop( #child ) = void then

    put "ERREUR (IsPlayerList):la liste de propriétés n'est pas correctement
structurée"
    return 0

end if

-- Cas où il n'y a pas les bons attributs dans la liste de propriétés
if lpPlayer.attributes.getaprop(#id) = void then

    put "ERREUR (IsPlayerList):la liste de propriétés n'a pas les bons
attributs"

```

```

    return 0
end if

iNbLevel = count( lpPlayer.child )

-- Parcours de tous les jeux du niveau
repeat with n=1 to iNbLevel

    -- Cas où la structure de la liste n'est pas valide
    if IsLevelList( lpPlayer.child[n] ) = 0 then

        return 0

    end if

end repeat

return 1
end IsPlayerList

-----
-- Fonction vérifiant la structure de la liste de propriétés dans sa globalité
-----

on IsProfileList( lpProfile )

    -- Cas où l'argument spécifié n'est pas une liste de propriétés
    if ilk(lpProfile) <> #proplist then

        put "ERREUR (IsProfileList): l'argument n'est pas une liste de propriétés"
        return 0

    end if

    -- Cas où la structure de la liste de propriétés est incorrecte
    if lpProfile.getaprop( #name ) = void or lpProfile.getaprop( #attributes ) =
void or lpProfile.getaprop( #child ) = void then

        put "ERREUR (IsProfileList): la liste de propriétés n'est pas correctement
structurée"
        return 0

    end if

    iNbPlayer = count( lpProfile.child )

    -- Parcours de tous les joueurs de la liste des profils
    repeat with n=1 to iNbPlayer

        -- Cas où la structure de la liste n'est pas valide
        if IsPlayerList( lpProfile.child[n] ) = 0 then

            return 0

        end if

    end repeat

    return 1
end IsProfileList

```

```
-----  
-- Fonction renvoyant la liste de tous les noms du profil le plus ancien au  
profil le plus récent --  
-----
```

```
on GetListProfile()
```

```
-----  
-- Initialisation des paramètres --  
-----
```

```
sFilePath = "profils.xml"
```

```
-----  
-- Chargement de la liste des profils --  
-----
```

```
sXml = ReadFile( sFilePath ) -- Lecture du fichier XML contenant les profils  
lpXml= ParseXml( sXml ) -- Conversion du contenu du fichier XML en liste de  
propriétés
```

```
-----  
-- Vérification de la structure des listes de propriétés --  
-----
```

```
-- Cas où la liste de propriétés des profils est invalide
```

```
if IsProfileList( lpXml ) = 0 then
```

```
    put "ERREUR (GetListProfile): liste de propriétés des profils invalide"  
    return 0
```

```
end if
```

```
iNbPlayer = count( lpXml.child )
```

```
-----  
-- Parcours de tous les profils --  
-----
```

```
repeat with n = 1 to iNbPlayer
```

```
    -- ajout de l'identifiant du profil à la liste  
    sList = sList &lpXml.child[n].attributes.id &RETURN
```

```
end repeat
```

```
return sList
```

```
end getListProfile
```

```
-----  
-- Fonction renvoyant la liste de tous les noms du profil le plus récent au  
profil le plus ancien --  
-----
```

```
on GetListProfileInv()
```

```
-----  
-- Initialisation des paramètres --  
-----
```

```

sFilePath = "profils.xml"

-----
-- Chargement de la liste des profils --
-----

sXml = ReadFile( sFilePath ) -- Lecture du fichier XML contenant les profils
lpXml= ParseXml( sXml ) -- Conversion du contenu du fichier XML en liste de
propriétés

-----
-- Vérification de la structure des listes de propriétés --
-----

-- Cas où la liste de propriétés des profils est invalide
if IsProfileList( lpXml ) = 0 then

    put "ERREUR (GetListProfileInv): liste de propriétés des profils invalide"
    return 0

end if

iNbPlayer = count( lpXml.child )

-----
-- Parcours de tous les profils --
-----
repeat with n = iNbPlayer down to 1

    -- ajout de l'identifiant du profil à la liste
    sList = sList &lpXml.child[n].attributes.id &RETURN

end repeat

return sList

end getListProfileInv

-----
-- Fonction renvoyant le nom du profil --
-----

on GetName( lpPlayer )

-----
-- Vérification de la liste de propriétés --
-----

-- cas où la liste spécifiée en argument n'est pas valide
if IsPlayerList( lpPlayer ) = 0 then

    put "ERREUR (GetName): liste invalide"
    return "ListeInvalide"

end if

return lpPlayer.attributes.id

end GetName

-----

```

```

-- Fonction renvoyant le score du niveau spécifié --
-----

on GetScore( lpPlayer, iLevelId)

-----
-- Vérification de la liste de propriétés --
-----

-- Cas où la liste spécifiée en argument n'est pas valide
if IsPlayerList( lpPlayer ) = 0 then

    put "ERREUR (GetScore): liste invalide"
    return 0

end if

iNbLevel = count( lpPlayer.child )

-- Cas où le niveau spécifié n'existe pas
if iLevelId > iNbLevel then

    put "ERREUR (GetScore): niveau invalide"
    return 0

end if

return lpPlayer.child[iLevelId].attributes.score
end GetScore

-----
-- Fonction mettant à jour le score du niveau spécifié --
-----

on UpdateScore( lpPlayer, iLevelId, iNewScore )

-----
-- Vérification de la liste de propriétés --
-----

-- Cas où la liste spécifiée en argument n'est pas valide
if IsPlayerList( lpPlayer ) = 0 then

    put "ERREUR (UpdateScore): liste invalide"
    return 0

end if

iNbLevel = count( lpPlayer.child )

-- Cas où le niveau spécifié n'existe pas
if iLevelId > iNbLevel then

    put "ERREUR (UpdateScore): niveau invalide"
    return 0

end if

-- Mise à jour du score
lpPlayer.child[iLevelId].attributes.score = string( iNewScore )

return 1

```

```
end updateScore
```

```
-----  
-- Fonction renvoyant le statut d'un jeu --  
-----
```

```
on GetStatus( lpPlayer, iGameId )
```

```
-----  
-- Vérification de la liste de propriétés --  
-----
```

```
-- Cas où la liste spécifiée en argument n'est pas valide
```

```
if IsPlayerList( lpPlayer ) = 0 then
```

```
    put "ERREUR (GetStatus): liste invalide"  
    return 0
```

```
end if
```

```
iNbLevel = count( lpPlayer.child )
```

```
-- Cas où le niveau spécifié n'existe pas
```

```
if iGameId > ( iNbLevel + 1)*10 then
```

```
    put "ERREUR (GetStatus): niveau invalide"  
    return 0
```

```
end if
```

```
-- Calcul du niveau de jeu
```

```
iLevelId = iGameId/10
```

```
iNbGame = count( lpPlayer.child[iLevelId].child )
```

```
-- Cas où le jeu spécifié n'existe pas
```

```
if iGameId > iNbGame+iLevelId*10 then
```

```
    put "ERREUR (GetStatus): jeu invalide"  
    return 0
```

```
end if
```

```
return lpPlayer.child[iLevelId].child[iGameId-iLevelId*10].attributes.statut
```

```
end GetStatus
```

```
-----  
-- Fonction mettant à jour le statut d'un jeu --  
-----
```

```
on UpdateStatus( lpPlayer, iGameId , iNewStatus)
```

```
-----  
-- Vérification de la liste de propriétés --  
-----
```

```
-- Cas où la liste spécifiée en argument n'est pas valide
```

```
if IsPlayerList( lpPlayer ) = 0 then
```

```
    put "ERREUR (UpdateStatus): liste invalide"  
    return 0
```

```

end if

iNbLevel = count( lpPlayer.child )

-- Cas où le niveau spécifié n'existe pas
if iGameId > ( iNbLevel + 1)*10 then

    put "ERREUR (UpdateStatus): niveau invalide"
    return 0

end if

-- Calcul du niveau de jeu
iLevelId = iGameId/10

iNbGame = count( lpPlayer.child[iLevelId].child )

-- Cas où le jeu spécifié n'existe pas
if iGameId > iNbGame+iLevelId*10 then

    put "ERREUR (UpdateStatus): jeu invalide"
    return 0

end if

-- Mise à jour du statut
lpPlayer.child[iLevelId].child[iGameId-iLevelId*10].attributes.statut = string
( iNewStatus )

return 1

end UpdateStatus

```

```

-----
-- Fonction renvoyant le nombre de parties faitent pour un jeu dans le niveau
spécifié --
-----

```

```

on GetNbParties( lpPlayer, iGameId )

-----
-- Vérification de la liste de propriétés --
-----

-- Cas où la liste spécifiée en argument n'est pas valide
if IsPlayerList( lpPlayer ) = 0 then

    put "ERREUR (GetNbParties): liste invalide"
    return 0

end if

iNbLevel = count( lpPlayer.child )

-- Cas où le niveau spécifié n'existe pas
if iGameId > ( iNbLevel + 1)*10 then

    put "ERREUR (GetNbParties): niveau invalide"
    return 0

end if

-- Calcul du niveau de jeu

```

```

iLevelId = iGameId/10

iNbGame = count( lpPlayer.child[iLevelId].child )

-- Cas où le jeu spécifié n'existe pas
if iGameId > iNbGame+iLevelId*10 then

    put "ERREUR (GetStatus): jeu invalide"
    return 0

end if

return lpPlayer.child[iLevelId].child[iGameId-iLevelId*10].
attributes.nbParties

end GetNbParties

-----
-- Fonction renvoyant le nombre de parties faitent pour un jeu dans le niveau
spécifié --
-----

on UpdateNbParties( lpPlayer, iGameId, iNewNbParties )

-----
-- Vérification de la liste de propriétés --
-----

-- Cas où la liste spécifiée en argument n'est pas valide
if IsPlayerList( lpPlayer ) = 0 then

    put "ERREUR (UpdateNbParties): liste invalide"
    return 0

end if

iNbLevel = count( lpPlayer.child )

-- Cas où le niveau spécifié n'existe pas
if iGameId > ( iNbLevel + 1)*10 then

    put "ERREUR (UpdateNbParties): niveau invalide"
    return 0

end if

-- Calcul du niveau de jeu
iLevelId = iGameId/10

iNbGame = count( lpPlayer.child[iLevelId].child )

-- Cas où le jeu spécifié n'existe pas
if iGameId > iNbGame+iLevelId*10 then

    put "ERREUR (UpdateStatus): jeu invalide"
    return 0

end if

-- Mise à jour du nombre de parties
lpPlayer.child[iLevelId].child[iGameId-iLevelId*10].attributes.nbParties =
string( iNewNbParties )

return 1

```

```
end UpdateNbParties
```

Chargement du profil

```
-----  
-- Chargement du profil --  
-----  
  
on click  
  
  sInputText = sprite(9).text  
  
  -- Cas où aucun nom de profil n'a été entré  
  if sInputText = "" then  
  
    alert( "Vous devez d'abord entrer un nom pour charger un profil." )  
  
    return 0  
  
  end if  
  
  -- Chargement du profil avec le nom spécifié  
  glpPlayer = LoadProfile( sInputText )  
  
  -- Cas où le profil a été trouvé  
  if glpPlayer <> 0 then  
  
    put "Chargement de la page de sélection des niveaux"  
    go to "niveaux"  
  
    return 1  
  
  end if  
  
  -- Cas où le profil n'a pas été trouvé  
  alert( "Impossible de trouver le profil " &sInputText &". " &RETURN &"Veuillez  
  créer un nouveau profil ou corriger votre saisie.")  
  
  return 0  
  
end click
```

Création d'un profil

```
-----  
-- Création d'un nouveau profil --  
-----  
  
on click  
  
  sInputText = sprite(9).text  
  
  -- Cas où aucun nom de profil n'a été entré  
  if sInputText = "" then  
  
    alert( "Vous devez d'abord entrer un nom pour créer un nouveau profil." )  
  
    return 0
```

```

end if

-----
-- Vérification que le profil n'existe pas déjà --
-----

-- Chargement du profil avec le nom spécifié
glpPlayer = LoadProfile( sInputText )

-- Cas où le profil a été trouvé
if glpPlayer <> 0 then

    alert( "Impossible de créer le profil " &sInputText &" parce ce nom est déjà
utilisé." &RETURN &"Veuillez essayer de nouveau avec un autre nom de profil.")

    return 0

end if

-- Cas où le profil n'a pas été trouvé

-----
-- Création du profil --
-----

-- Création du profil avec le nom spécifié
glpPlayer = createProfile( sInputText )

-- Cas où le profil a pu être créée
if glpPlayer <> 0 then

    put "Création d'un nouveau profil"
    go to "creer"

    return 1

end if

-- Cas où le profil n'a pas pu être créée
alert( "Impossible de créer le profil " &sInputText &". " &RETURN &"Veuillez
vérifier l'espace libre disponible et les droits d'accès du support.")

return 0

end click

```

JEU DES PHRASES A RECONSTITUER

Extraits du fichier jeu1.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<jeu>
  <niveau id="1">
    <phrase id="1" valeur="Le facteur distribue le courrier.">
      <mot valeur="Le" position="1" />
      <mot valeur="facteur" position="2" />
      <mot valeur="distribue" position="3" />
      <mot valeur="le" position="4" />
      <mot valeur="courrier." position="5" />
    </phrase>
    <phrase id="2" valeur="Le sportif s'entraîne au stade.">
      <mot valeur="Le" position="1" />
      <mot valeur="sportif" position="2" />
      <mot valeur="s'entraîne" position="3" />
      <mot valeur="au" position="4" />
      <mot valeur="stade." position="5" />
    </phrase>
  </niveau>

  <niveau id="2">
    <phrase id="1" valeur="Pour aller danser, je mettrai ma jolie robe rouge.">
      <mot valeur="Pour" position="1" />
      <mot valeur="aller" position="2" />
      <mot valeur="danser," position="3" />
      <mot valeur="je" position="4" />
      <mot valeur="mettrai" position="5" />
      <mot valeur="ma" position="6" />
      <mot valeur="jolie" position="7" />
      <mot valeur="robe" position="8" />
      <mot valeur="rouge." position="9" />
    </phrase>
    <phrase id="2" valeur="Le match de football aura lieu au stade municipal en fin d'après-midi.">
      <mot valeur="Le" position="1" />
      <mot valeur="match" position="2" />
      <mot valeur="de" position="3" />
      <mot valeur="football" position="4" />
      <mot valeur="aura" position="5" />
      <mot valeur="lieu" position="6" />
      <mot valeur="au" position="7" />
      <mot valeur="stade" position="8" />
      <mot valeur="municipal" position="9" />
      <mot valeur="en" position="10" />
      <mot valeur="fin" position="11" />
      <mot valeur="d'" position="12" />
      <mot valeur="après-midi." position="13" />
    </phrase>
  </niveau>
</jeu>
```

Gestion des phrases

```
-----
-- Fonction chargeant sous la forme d'une liste de propriétés la phrase spécifiée
du niveau correspondant --
-----
```

```

on LoadGameLevel( iLevelId )

-----
-- Initialisation des paramètres --
-----

sFilePath = "jeu1.xml"

-----
-- Chargement de la liste des profils --
-----

sXml = ReadFile( sFilePath ) -- Lecture du fichier XML contenant les profils

lpXml= ParseXml( sXml ) -- Conversion du contenu du fichier XML en liste de
propriétés

-----
-- Vérification de la structure des listes de propriétés --
-----

-- Cas où la liste de propriétés des profils est invalide
if IsGameList( lpXml ) = 0 then

    put "ERREUR (LoadGameLevel): liste de propriétés des profils invalide"
    return 0

end if

iNbLevel = count( lpXml.child )

-----
-- Recherche du niveau --
-----

repeat with n = 1 to iNbLevel

    -- Cas où le niveau existe
    if lpXml.child[n].attributes.id = iLevelId then

        put "Chargement du niveau " &iLevelId

        return lpXml.child[n] -- On renvoie la partie de la liste correspondant au
niveau recherché

    end if

end repeat

-- Cas où le niveau n'existe pas
put "ERREUR (LoadGameLevel): impossible de charger le niveau " &iLevelId

return 0

end LoadGameLevel

-----

-- Fonction renvoyant sous forme de liste de propriétés la phrase spécifiée du
niveau correspondant --
-----

on LoadSentence( lpLevel, iSentenceId )

-----

```

```

-- Vérification de la liste de propriétés --
-----

-- cas où la liste spécifiée en argument n'est pas valide
if IsLevelList( lpLevel ) = 0 then

    put "ERREUR (LoadSentence): liste invalide"
    return "ListeInvalide"

end if

iNbSentence = count( lpLevel.child )

-- Cas où la phrase spécifiée n'existe pas
if iSentenceId > iNbSentence then

    put "ERREUR (LoadSentence): phrase invalide"
    return "ListeInvalide"

end if

return lpLevel.child[iSentenceId]

end LoadSentence

-----
-- Fonction vérifiant la structure de la liste de propriétés au niveau du mot
-----

on IsWordList( lpWord )

-- Cas où l'argument spécifié n'est pas une liste de propriétés
if ilk(lpWord) <> #proplist then

    put "ERREUR (IsWordList): l'argument n'est pas une liste de propriétés"
    return 0

end if

-- Cas où la structure de la liste de propriétés est incorrecte
if lpWord.getaprop( #name ) = void or lpWord.getaprop( #attributes ) = void or
lpWord.getaprop( #child ) = void then

    put "ERREUR (IsWordList):la liste de propriétés n'est pas correctement
structurée"
    return 0

end if

-- Cas où il n'y a pas les bons attributs dans la liste de propriétés
if lpWord.attributes.getaprop( #valeur ) = void or lpWord.attributes.getaprop(
#position ) = void then

    put "ERREUR (IsWordList):la liste de propriétés n'a pas les bons attributs"
    return 0

end if

return 1

end IsWordList

```

```
-----  
-- Fonction vérifiant la structure de la liste de propriétés au niveau du niveau  
de la phrase jeu --  
-----
```

```
on IsSentenceList( lpSentence )
```

```
-- Cas où l'argument spécifié n'est pas une liste de propriétés
```

```
if ilk(lpSentence) <> #proplist then
```

```
    put "ERREUR (IsSentenceList): l'argument n'est pas une liste de propriétés"  
    return 0
```

```
end if
```

```
-- Cas où la structure de la liste de propriétés est incorrecte
```

```
if lpSentence.getaprop( #name ) = void or lpSentence.getaprop( #attributes ) =  
void or lpSentence.getaprop( #child ) = void then
```

```
    put "ERREUR (IsSentenceList):la liste de propriétés n'est pas correctement  
structurée"  
    return 0
```

```
end if
```

```
-- Cas où il n'y a pas les bons attributs dans la liste de propriétés
```

```
if lpSentence.attributes.getaprop( #id ) = void or  
lpSentence.attributes.getaprop( #valeur ) = void then
```

```
    put "ERREUR (IsSentenceList):la liste de propriétés n'a pas les bons  
attributs"  
    return 0
```

```
end if
```

```
iNbWord = count( lpSentence.child )
```

```
-- Parcours de tous les mots d'une phrase
```

```
repeat with n = 1 to iNbWord
```

```
-- Cas où la structure de la liste n'est pas valide
```

```
if IsWordList( lpSentence.child[n] ) = 0 then
```

```
    return 0
```

```
end if
```

```
end repeat
```

```
return 1
```

```
end IsSentenceList
```

```
-----  
-- Fonction vérifiant la structure de la liste de propriétés au niveau du niveau  
de jeu --  
-----
```

```
on IsLevelList( lpLevel )
```

```
-- Cas où l'argument spécifié n'est pas une liste de propriétés
```

```
if ilk(lpLevel) <> #proplist then
```

```

    put "ERREUR (IsLevelList): l'argument n'est pas une liste de propriétés"
    return 0

end if

-- Cas où la structure de la liste de propriétés est incorrecte
if lpLevel.getaprop( #name ) = void or lpLevel.getaprop( #attributes ) = void
or lpLevel.getaprop( #child ) = void then

    put "ERREUR (IsLevelList):la liste de propriétés n'est pas correctement
structurée"
    return 0

end if

-- Cas où il n'y a pas les bons attributs dans la liste de propriétés
if lpLevel.attributes.getaprop(#id) = void then

    put "ERREUR (IsLevelList):la liste de propriétés n'a pas les bons attributs"
    return 0

end if

iNbSentence = count( lpLevel.child )

-- Parcours de tous les niveaux du jeu
repeat with n=1 to iNbSentence

    -- Cas où la structure de la liste n'est pas valide
    if IsSentenceList( lpLevel.child[n] ) = 0 then

        return 0

    end if

end repeat

return 1

end IsLevelList

```

```

-----
-- Fonction vérifiant la structure de la liste de propriétés dans sa globalité
-----

```

```

on IsGameList( lpGame )

    -- Cas où l'argument spécifié n'est pas une liste de propriétés
    if ilk(lpGame) <> #proplist then

        put "ERREUR (IsGameList): l'argument n'est pas une liste de propriétés"
        return 0

    end if

    -- Cas où la structure de la liste de propriétés est incorrecte
    if lpGame.getaprop( #name ) = void or lpGame.getaprop( #attributes ) = void or
lpGame.getaprop( #child ) = void then

        put "ERREUR (IsGameList):la liste de propriétés n'est pas correctement
structurée"
        return 0

    end if

end IsGameList

```

```

end if

iNbLevel = count( lpGame.child )

-- Parcourt de tous les joueur de la liste des profils
repeat with n=1 to iNbLevel

    -- Cas où la structure de la liste n'est pas valide
    if IsLevelList( lpGame.child[n] ) = 0 then

        return 0

    end if

end repeat

return 1

end IsGameList

-----
-- Fonction renvoyant la valeur d'une phrase --
-----

on GetSentence( lpSentence )

    -----
    -- Vérification de la liste de propriétés --
    -----

    -- Cas où la liste spécifiée en argument n'est pas valide
    if IsSentenceList( lpSentence ) = 0 then

        put "ERREUR (GetSentence): liste invalide"
        return "ListeInvalide"

    end if

    return lpSentence.attributes.valeur

end GetSentence

-----
-- Fonction renvoyant la valeur du mot spécifié --
-----

on GetWord( lpSentence, iPosition)

    -----
    -- Vérification de la liste de propriétés --
    -----

    -- Cas où la liste spécifiée en argument n'est pas valide
    if IsSentenceList( lpSentence ) = 0 then

        put "ERREUR (GetWord): liste invalide"
        return "ListeInvalide"

    end if

    iNbWord = count( lpSentence.child )

```

```

-- Cas où le mot spécifié n'existe pas
if iPosition > iNbWord then

    put "ERREUR (GetWord): position invalide"
    return "PositionInvalide"

end if

repeat with n=1 to iNbWord

    -- Cas où le mot a été trouvé
    if lpSentence.child[n].attributes.position = iPosition then

        return lpSentence.child[n].attributes.valeur -- On renvoie la valeur du
mot pour la position spécifiée

    end if

end repeat

-- Cas où le mot n'a pas été trouvé
put "ERREUR (GetWord): position invalide"
return "PositionInvalide"

end GetWord

```

```

-----
-- Fonction renvoyant une liste contenant un nombre voulu d'entiers aléatoires
compris dans les bornes spécifiées --
-----

```

```

on GetRandomList( iMin, iMax, iNumber )

    -----
    -- Vérification des paramètres --
    -----

    if iNumber > iMax then

        put "ERREUR (GetRandomList): écart trop petit pour générer le nombre
d'entiers souhaités"
        return 0

    end if

    -----
    -- Génération de la liste de nombres aléatoires --
    -----

    lRandomList = []

    iNbRandom = 0

    repeat while iNbRandom < iNumber

        -- Génération d'un nombre aléatoire compris entre les bornes spécifiées
iCurrentRandom = random( iMin, iMax)

        -- cas où le nombre généré aléatoirement n'est pas déjà présent dans la
liste
        if lRandomList.getPos( iCurrentRandom ) = 0 then

```

```

-- on ajoute le nombre aléatoire à la liste
lRandomList.add( iCurrentRandom )

iNbRandom = iNbRandom+1

end if

end repeat

return lRandomList

end GetRandomList

```

Initialisation du jeu

```

-----
-- Définition des variables globales --
-----

global glpSentenceList -- Liste de toutes les phrases du niveau courant
global glpCurrentSentence -- Phrase courante

global glRandomSentenceList -- Liste des indices de phrases à charger
aléatoirement
global giNbTotalSentence -- Nombre de phrases à retrouver pour l'étape courante
global giNbFindSentence -- Nombre de phrases trouvées
global giNbCurrentSentence -- Nombre de phrases tentées

global giNbTotalTry -- Nombre total d'essais permis sur une même phrase
global giNbCurrentTry -- Nombre d'essais effectués sur la phrase courante

global giCurrentSprite -- Prochain emplacement disponible

global glActionList -- Liste des actions effectuées

-----
-- Initialisation paramètres du jeu --
-----

on exitFrame me

-- Définition des paramètres du cadre contenant les mots
iBeginX = 125
iEndX = 754
iBeginY = 95
iEndY = 442

-- Initialisation des variables courantes
iLastX = 0
iLastY = 0
iLastWidth = 0

-- Initialisation du champs recevant la phrase à reconstituer dans l'ordre
member( "phrase" ).text = ""

-- Initialisation des compteurs
member( "compteurPhrase" ).text = string( giNbCurrentSentence )
member( "nbPhrase" ).text = string( giNbTotalSentence )
member( "compteurEssai" ).text = string( giNbCurrentTry )
member( "nbEssai" ).text = string( giNbTotalTry )

-- Calcul du nombre de mots de la phrase

```

```

iNumberWord = count( glpCurrentSentence.child )

-- Génération de la liste de chargement aléatoire des mots
lRandomWordList = getRandomList( 1, iNumberWord, iNumberWord )

put lRandomWordList

-- Parcours de tous les mots de la phrase
repeat with n=1 to iNumberWord

    -- Création d'un nouvel acteur texte de la distribution qui contient le mot
    courant
    sCurrentWord = glpCurrentSentence.child[ lRandomWordList[n] ].
attributes.valeur

    aText = new( #text )
    member(aText).font = "GillSans" -- Définition de la police
    member(aText).fontSize = 30 -- Définition de la taille de la police du texte
    member(aText).text = sCurrentWord -- Définition du texte
    member(aText).width = sCurrentWord.char.count*18 -- Dimensionnement selon le
texte

-----
-- Calcul de la position de l'acteur dynamique --
-----

iCurrentWidth = member(aText).width

-- Cas où aucun mot n'a été crée
if n = 1 then

    -- Positionnement du premier mot au début de la zone
    iPositionX = iBeginX+15
    iPositionY = iBeginY+15

    -- Cas où au moins un mot a déjà été crée
else

    iPositionX = iLastX+iLastWidth+40
    iCurrentEndX = iPositionX+member(aText).width

    -- Cas où le mot ne tient pas sur la ligne
    if iCurrentEndX > iEndX then

        iPositionX = iBeginX+15
        ipositionY = iLastY+45

        -- Cas où le mot tient sur la ligne
    else

        iPositionY = iLastY

    end if
end if

-- Enregistrement des valeurs du mot courant pour le traitement suivant
iLastX = iPositionX
iLastY = iPositionY
iLastWidth = iCurrentWidth

-- Définition des paramètres du mot
lpWordParam = propList()
lpWordParam[#member] = aText
lpWordParam[#x] = iPositionX
lpWordParam[#y] = ipositionY

```

```

lpWordParam[#channel] = giCurrentSprite

-- Création d'un nouvel objet mot
script("ObjetMot").new(lpWordParam)

giCurrentSprite = giCurrentSprite + 1 -- Incrémentation pour le sprite
suivant

end repeat
end exitFrame

```

Création des objets constituant les mots

```

-----
-- Définition des variables globales --
-----

global glpSentenceList -- Liste de toutes les phrases du niveau courant
global glpCurrentSentence -- Phrase courante

global glRandomSentenceList -- Liste des indices de phrases à charger
aléatoirement
global giNbTotalSentence -- Nombre de phrases à retrouver pour l'étape courante
global giNbFindSentence -- Nombre de phrases trouvées
global giNbCurrentSentence -- Nombre de phrases tentées

global giNbTotalTry -- Nombre total d'essais permis sur une même phrase
global giNbCurrentTry -- Nombre d'essais effectués sur la phrase courante

global giCurrentSprite -- Prochain emplacement disponible

global glActionList -- Liste des actions effectuées

-----
-- Définition des propriétés de l'instance --
-----

property pspWord -- mot courant
property iPush -- champs sélectionné ou pas

-----
-- Constructeur d'un objet constituant un mot --
-----

on new ( me, lpParams )

iPush = 0

-- Cas où le paramètre n'est pas une liste de propriétés
if ilk( lpParams ) <> #proplist then

    put "ERREUR (GestionMot): propriétés de l'objet mot non définis"
    exit

end if

iSpriteNumber = lpParams[#channel]
iX = lpParams[#x]

```

```

iY = lpParams[#y]
mDistribution = lpParams[#member]

-- Vérification des paramètres
if not integerP( iSpriteNumber ) then

    put "ERREUR (GestionMot): position de sprite invalide"
    exit

end if

if not integerP( iX ) then

    put "ERREUR (GestionMot): Coordonnée en X invalide"
    exit

end if

if not integerP( iY ) then

    put "ERREUR (GestionMot): Coordonnée en Y invalide"
    exit

end if

if not ilk( mDistribution, #member ) then

    put "ERREUR (GestionMot): membre de la distribution incorrect"
    exit

end if

-----
-- Création du sprite --
-----

pspWord = sprite( iSpriteNumber )
puppetSprite( iSpriteNumber, TRUE ) -- Contrôle du sprite par un script
pspWord.loc = point( iX, iY )
pspWord.member = mDistribution

-- Ajout dynamique de comportements à l'objet
(the actorList).add(me)
pspWord.scriptInstanceList.add(me)

return me

end new

-----
-- Fonction de suppression de l'instance de l'objet --
-----

on kill me

-- Suppression de l'acteur dans la distribution
pspWord.member.erase()

-- Suppression des comportements
pspWord.scriptInstanceList.deleteOne(me)
(the actorList).deleteOne(me)

-- Suppression de l'objet
iSpriteNumber = pspWord.spriteNum

```

```

    pspWord.member = VOID
    puppetSprite iSpriteNumber, FALSE
end kill

-----
-- Fonction de réinitialisation de l'objet --
-----

on reset me

    pspWord.blend = 100
    iPush = 0

end reset

-----
-- Fonction pour cacher l'objet --
-----

on hide me

    pspWord.visible = false

end hide

-----
-- Fonction pour afficher l'objet --
-----

on show me

    pspWord.visible = true

end hide

-----
-- Action de sélection du mot --
-----

on mouseUp me

    -- Cas où le bouton n'a pas déjà été sélectionné
    if iPush = 0 then

        -- Ajout du mot à la liste des actions
        lpWord = [#id:pspWord.spriteNum, #valeur:pspWord.member.text]
        glActionList.add( lpWord )

        -- Cas où la phrase de réponse ne contient aucun mot
        if member( "phrase" ).text = "" then

            member( "phrase" ).text = pspWord.member.text

```

```

    -- Cas où la phrase comporte au moins un mot
else
    member( "phrase" ).text = member( "phrase" ).text & " "
&pspWord.member.text

end if

-- Modification de l'acteur pour indiquer qu'il a déjà été sélectionné
pspWord.blend = 50
iPush = 1

-- Cas où le bouton a déjà été sélectionné
else

-- Remise à zéro du champs de réponse
member( "phrase" ).text = ""

-- Reconstruction de la phrase de réponse
repeat with n=1 to glActionList.count

-- Cas où l'on est sur le mot courant à enlever
if glActionList[n].id = pspWord.spriteNum then

    iWordToDelete = n

else

-- Cas du premier mot de la phrase
if n = 1 then

    member( "phrase" ).text = glActionList[n].valeur

else

-- Cas où l'on supprime le 1er mot de la phrase et qu'il faut ne pas
mettre d'espace pour le 2ème mot
if iWordToDelete = 1 and n = 2 then

    member( "phrase" ).text = glActionList[n].valeur

else

    member( "phrase" ).text = member( "phrase" ).text & " " &glActionList
[n].valeur

end if

end if

end repeat

-- Suppression du mot courant de la liste de l'historique
glActionList.deleteAt( iWordToDelete )

-- Modification de l'acteur pour indiquer qu'il n'est plus sélectionné
pspWord.blend = 100
iPush = 0

end if

end mouseUp

```

JEU DE LA CORRECTION SNTAXIQUE

Version partielle du fichier XML jeu2.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<phrases>
  <etape id="1">
    <phrase debut="" incorrect="Le" correct="La" fin=" fille joue à la poupée."
    niveau="1" />
    <phrase debut="La maîtresse donne " incorrect="un" correct="une" fin="
    feuille à chaque élève." niveau="1" />
    <phrase debut="" incorrect="Sa" correct="Son" fin=" père se rase."
    niveau="1" />
    <phrase debut="Pose " incorrect="ton" correct="ta" fin=" trousse sur la
    table." niveau="1" />
    <phrase debut="Je préfère " incorrect="la" correct="le" fin=" citron à la
    fraise." niveau="1" />
    <phrase debut="Elle souhaite bonne année " incorrect="à la" correct="au"
    fin=" directeur." niveau="1" />

    <phrase debut="Louis a peur de monter " incorrect="dans" correct="sur"
    fin=" le manège." niveau="2" />
    <phrase debut="En jouant, des billes ont roulé " incorrect="sur"
    correct="sous" fin=" le lit." niveau="2" />
    <phrase debut="Je pars en vacances " incorrect="dans" correct="à" fin=" la
    montagne." niveau="2" />
    <phrase debut="Je " incorrect="m'ai" correct="me suis" fin=" fait mal au
    genou." niveau="2" />
    <phrase debut="Le clown " incorrect="sont" correct="est" fin =" tombé sur
    le nez." niveau="2" />
    <phrase debut="Chaque soir, Paul et Lucie " incorrect="fait" correct="font"
    fin=" leur toilette." niveau="2" />

  </etape>
  <etape id="2">
    <phrase debut="Regarde l'oiseau qui vole au " incorrect="dessous"
    correct="dessus" fin=" de la maison." niveau="1" />
    <phrase debut="Pioche la carte posée sur le " incorrect="dessous"
    correct="dessus" fin=" du tas." niveau="1" />
    <phrase debut="On habite au rez-de-chaussée, on n'a pas de voisin en "
    incorrect="dessus" correct="dessous" fin="." niveau="1" />
    <phrase debut="Au dernier étage, personne ne nous dérange " incorrect="en
    dessous" correct="au dessus" fin="." niveau="1" />
    <phrase debut="Elle revient " incorrect="à" correct="de" fin=" la
    boulangerie, avec une baguette." niveau="1" />

    <phrase debut="Passe " incorrect="sur" correct="par" fin=" ce raccourci,
    c'est plus court." niveau="2" />
    <phrase debut="Ne te mets pas " incorrect="derrière" correct="devant" fin="
    moi, je ne vois plus rien." niveau="2" />
    <phrase debut="L'ogre est assis " incorrect="avant" correct="devant" fin="
    un très gros repas." niveau="2" />
    <phrase debut="Cache toi " incorrect="devant" correct="derrière" fin="
    l'arbre." niveau="2" />
    <phrase debut="Ses yeux brillent " incorrect="devant" correct="derrière"
    fin=" ses lunettes." niveau="2" />

  </etape>
</phrases>
```

Initialisation des variables pour le niveau 1

```
fait = new Array();

// Pour résoudre les problèmes d'accents...
system.useCodepage = true;

// Initialisation des variables
score_tmp=0;
essai = 1;
phra = 1;
lvl = 1;
nb_essais = 2;
nb_phrases = 5;
mc_aide._alpha = 0;

phrases = phra+"/"+nb_phrases;
score_max=(nb_essais*nb_phrases);

gotoAndStop(10);
```

Parsage du fichier XML, sélection aléatoire puis affichage d'une phrase

```
//Placement du champ dynamique
txtdyn_phrase._x = 150;
txtdyn_phrase._y = 250;

// Debut du script de récupération des données XML
classe = new XML();
classe.ignoreWhite = true;

// Curseur ecriture dans la zone de texte
focusManager.setFocus(lbl_reponse);

// Pour que le curseur main ne s'affiche pas sur la faute
btn_incorrect.useHandCursor = false;

classe.onLoad = function(etatCharge) {
    if (etatCharge) {

        alea = "";

        // Calcul du nombre de noeuds XML
        nb_noeuds = this.firstChild.childNodes[0].childNodes.length-1;

        //Choix au hasard d'une phrase
        if (!alea) { // Au début du jeu
            alea = Math.round(Math.random()*nb_noeuds);
        }
        else if (alea = ""){ // Après une réponse correcte
            alea = Math.round(Math.random()*nb_noeuds);
        }

        // Vérification que la phrase n'est pas encore passée
        j = 0;

        while (j<(fait.length)) {
            if (fait[j] != alea) {
                ok = " oui";
            }
            else {
                ok = "non";
            }
        }
    }
}
```

```

        break;
    }
    j++;
}

if (ok == "non") gotoAndPlay(25);

else {

// Sélection de la phrase dans le fichier XML
noeuds = this.firstChild.childNodes[0].childNodes[alea];

affichage(noeuds);

// Si le niveau n'est pas bon, on reprend une autre phrase
if (niveau != lvl) gotoAndPlay(25);

// Calcul du placement du bouton et gestion de sa transparence
placementX_char=debut.length;
placementX_px=placementX_char*9;
taille_char=incorrect.length;
taille=taille_char*9;

btn_incorrect._x = txtdyn_phrase._x+placementX_px;
btn_incorrect._y = txtdyn_phrase._y;
btn_incorrect._width = taille;
if (!alpha) {
    alpha = 0;
}
btn_incorrect._alpha = alpha;

}

};

// Désigne le fichier XML appelé
classe.load("jeu2.xml");

// Fonction récupérant les données dans le fichier XML
function affichage(noeud) {
    debut = noeud.attributes.debut;
    incorrect = noeud.attributes.incorrect;
    correct = noeud.attributes.correct;
    fin = noeud.attributes.fin;
    niveau = noeud.attributes.niveau;

    phrase = debut+incorrect+fin;
}

```

Bouton de validation après complétion du champ de texte éditable

```

on (release) {
    reponse = lbl_reponse.text;
    gotoAndStop(12);
}

```

Vérification de la réponse puis traitement en fonction du résultat

```

if (reponse == correct) {
    gotoAndStop(14);
}

```

```

}
else {
    alpha = 30;
    phrase_juste = debut+"<u>" + correct + "</u>" + fin;
    phrase = debut + incorrect + fin;
    essai++;
    if (essai > nb_essais) {
        gotoAndPlay(18);
    }
    else {
        essais = essai + "/" + nb_essais;
        gotoAndStop(11);
    }
}
}

```

Réinitialisation des variables et contrôle de l'avancement du jeu

```

alpha = 0;
score_tmp = score_tmp + (nb_essais - (essai - 1));
essai = 1;
essais = essai + "/" + nb_essais;
phra++;
if (phra > nb_phrases) {
    gotoAndPlay(30);
}
else {
    fait.push(alea);
    phrases = phra + "/" + nb_phrases;
    gotoAndStop(10);
}
}

```

Calcul du score

```

stop();
iProgress = Math.round((score_tmp / score_max) * 100);
geturl(waitflash);

```

Bouton d'affichage de l'aide

```

on(press) {
    if (mc_aide._alpha == 0) {
        mc_aide._alpha = 100;
    }
    else if (mc_aide._alpha == 100) {
        mc_aide._alpha = 0;
    }
}
}

```

JEU DE DISCRIMINATION DES PHONEMES AUDITIVEMENT PROCHES

Un fichier XML codant le niveau 2

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<sons>

  <categorie nom="pb">

    <son num="1" fichier="sons/niveau3/saparézovi.mp3" cat="pb" choix1="p"
choix2="b"      ok="p"/>
    <son num="2" fichier="sons/niveau3/rontupégafgré.mp3" cat="pb" choix1="p"
choix2="b"      ok="p"/>
    <son num="3" fichier="sons/niveau3/fankévobuja.mp3" cat="pb" choix1="p"
choix2="b"      ok="b"/>
    <son num="4" fichier="sons/niveau3/chibédalomé.mp3" cat="pb" choix1="p"
choix2="b"      ok="b"/>
    <son num="5" fichier="sons/niveau3/cantébalidèr.mp3" cat="pb" choix1="p"
choix2="b"      ok="b"/>
    <son num="6" fichier="sons/niveau3/ronémouponsi.mp3" cat="pb" choix1="p"
choix2="b"      ok="p"/>

  </categorie>

  <categorie nom="dt">

    <son num="1" fichier="sons/niveau3/chalémivuté.mp3" cat="dt" choix1="t"
choix2="d"      ok="t"/>
    <son num="2" fichier="sons/niveau3/prouchumalodin.mp3" cat="dt" choix1="t"
choix2="d"      ok="d"/>
    <son num="3" fichier="sons/niveau3/fridouvimidin.mp3" cat="dt" choix1="t"
choix2="d"      ok="d"/>
    <son num="4" fichier="sons/niveau3/jingalopétu.mp3" cat="dt" choix1="t"
choix2="d"      ok="t"/>

  </categorie>

</sons>
```

Chargement du XML et sélection du nœud de catégorie X

```
// Chargement XML et initialisation
son_xml = new XML();
son_xml.ignoreWhite = true;
son_xml.onLoad = function(ok) {

  if (ok) {
    //recherche dans les noeuds de la catégorie choisie
    for (var i=0;i<son_xml.firstChild.childNodes.length;i++) {

      //noeud de la categorie actuelle
      if (choixCategorie == son_xml.firstChild.childNodes[i].
attributes.nom) {

        //noeuds enfants de la categorie actuelle
        noeuds = son_xml.firstChild.childNodes[i].childNodes;

      }
    }
  }
}
```

```

        // on récupère le nombre de noeuds à ce niveau
        nbrsons = noeuds.length;
        // on sélectionne le premier noeud
        premierson = noeuds[0];
        // on sélectionne le dernier noeud
        dernierson = noeuds[nbrsons-1];
        // enCours correspond au noeud en cours d'affichage
        enCours = premierson;
        Afficheson(enCours);
    }
};

//chargement du fichier XML
son_xml.load("XML/jeu31.xml");

// lecture du son
function Lectureson(son) {
    tmp = new Sound();
    tmp.loadSound( son.attributes.fichier , true);
    tmp.start();
    choix1 = son.attributes.choix1;
    choix2 = son.attributes.choix2;
    compteur = son.attributes.num + "/" + nbrsons;
}

set (coups,1);//variable comptant le nombre de coups joués
set (score,0);//variable contenant le ratio du score
set(ratio,10);//variable définissant le ratio à appliquer au score
set (essai,0);//variable pour indiquer le nombre d'essai tenté par le joueur
pour un son

essai = 0;

// Affichage des choix
function Afficheson(son) {
    choix1 = son.attributes.choix1;
    choix2 = son.attributes.choix2;
}

//fontion qui retourne le score dans une variable
function getScore(score) {
    return score;
}

```

Exemple du code d'un bouton de choix de possibilités

```

// Gestion du bouton gauche
on (release)
{
    //si c'est la bonne réponse :
    if (enCours.attributes.ok == choix1) {
        //lancement du bravo
        gotoAndPlay("bravo");
    }

    //sinon relecture
    else {
        //retour au début
        gotoAndPlay("recommence");
    }
}

```

Code du bouton qui lance le son

```
// Gestion du bouton son
on (release) {
    Lectureson(enCours);
    //activation des boutons de choix

    gauche.enabled = true;
    droit.enabled = true;
}
```

Code lorsque le joueur n'a pas réussi

```
//re-écoute du son
Lectureson(enCours);

//activation du bouton de son
bouton_son.enabled = true;

//mise à jour de la variable essai
essai = 1;

//retour au début
gotoAndPlay(2);
```

Code du script lorsque le joueur a réussi

```
//re-écoute du son
Lectureson(enCours);

//changement de son
if (coups <= 10) {
    //activation du bouton son
    bouton_son.enabled = true;

    //passage au son suivant dans le XML
    enCours = enCours.nextSibling;

    //incrémentement du nombre de coups joués
    coups ++;

    //calcul du score
    if (essai == 1) {
        score += ratio/2;
    }

    else score += ratio;

    //retour jeu
    essai = 0;
    gotoAndPlay(2);
}

//si on est à la fin
else if (coups == 10) {
    //le score passé dans la variable iprogress
    iProgress = getScore(score);
    //qui est recherchée par le script director parent au lancement de
    « waitflash »
    getUrl("waitFlash", GET);
}
```