

Documentation Code : VivreOuSurvivre

Développeur : Anthony CARDOSO-MOREIRA & El-Khair NOURDINE

Présentation :

- **VivreOuSurvivre : Késako ?**

VivreOuSurvivre est un jeu ludo-pédagogique conçu pour initier les enfants aux bases des algorithmes. Le joueur évolue sur un plateau avec une carte générée aléatoirement, et son objectif est d'atteindre les drapeaux tout en accumulant un maximum de réussites.

- **Quels sont les règlements du jeu ?**

Le joueur devra atteindre le drapeau en utilisant un algorithme, en s'aidant d'un ensemble d'outils mis à sa disposition (avancer, reculer, droite, gauche, boucle "pour", boucle "tant que"). Cependant, si le drapeau n'est pas atteint à la fin de son algorithme, le joueur devra recommencer depuis le début de la carte. À noter que les bombes explosées ou autres éléments de la carte ne réapparaîtront pas.

Voici les règlements du jeu **VivreOuSurvivre** :

1. **Point de Vie (PV)** : Le joueur commence avec un total de 10 ❤️ .
2. **Lave** : Si le joueur touche la lave "🔥", il perd 5 de ses PV.
3. **Bombe** : Si le joueur touche une bombe "💣", il perd 1 PV.
4. **Arbre** : Si le joueur essaie de traverser un arbre "🌳", il a 20 % de chances de prendre des dégâts.
5. **Montagne** : Le joueur ne peut pas traverser la montagne "🏔️".
6. **Pertes de PV** : Lorsque le joueur perd des PV, Kaomiji s'énerve. Si les PV du joueur atteignent 0, soit "💔", Kaomiji abattra le joueur.
7. **Cartes événements** : Des cartes événements "🎴" peuvent donner des **bonus** ou des **malus** au joueur.
8. **Bouclier** : Le bouclier protège le joueur contre certains dégâts, agissant comme un deuxième PV.
9. **Immunité** : L'immunité contre les malus n'est pas stackable.



Description du code :

- **Création de type :**

```
Joueur newJoueur();
```

Le type **Joueur** représente un joueur que l'on crée pour commencer un jeu. Le joueur choisit un nom (**String**), un genre (**String**) et un personnage (**String**).

Le **tutoriel** (**boolean**) permet de savoir si le joueur est dans le tutoriel, car le jeu est plus tolérant dans cette partie.

Le **nbBouclier** (**int**) est un bonus que le joueur peut obtenir dans les cartes événements, tout comme l'**immunité** (**boolean**).

Pour localiser le joueur sur la carte, **idxL** (**int**) et **idxC** (**int**) indiquent sa position dans le tableau à deux dimensions. Ces indices seront utilisés pour déplacer le joueur ou afficher sa **position** (**String**).

```
Objectif newObjectif();
```

Le type **Objectif**, c'est le drapeau rouge "▶" que le joueur doit atteindre pour gagner des **nbReussite**. On veut le créer pour le rendre plus facile à trouver sur la carte, et pour savoir si le joueur a atteint le drapeau rouge ou pas. Elle est composée du drapeau rouge (**String**), ainsi que son indice de ligne (**int**) et de colonne (**int**).

- **Kaomiji, le maître du jeu :**

```
int minimum(int premierNb, int deuxiemeNb);
```

Cette fonction retournera le plus petit nombre entre les deux et est surtout utilisée dans la fonction **maitreKaomiji()** pour déterminer et enregistrer la case du tableau où se trouvent les expressions de Kaomiji, en fonction du nombre de PV du joueur.

```
String maitreKaomiji(int nbChances);
```

Cette fonction retournera l'état de Kaomiji. Elle sera adorable si les PV du joueur sont égaux à 10. Cependant, plus les PV du joueur baisseront, plus Kaomiji deviendra sauvage et sans pitié.

```
String espacement(String mot);
```

Cette fonction retournera un espace de la taille du **mot**, ce qui permettra d'aligner les mots et les phrases quand Kaomiji parlera.

```
String kaomijiPhrase(String mot);
```

Cette fonction, principalement utilisée dans `kaomijiOrateur()` et `kaomijiOrateurln()`, retournera l'état de Kaomiji suivi du `mot` qui sera mis en paramètre.

```
void kaomijiOrateur(String mot);
```

Cette procédure est l'équivalent de `print()` et affichera l'état de Kaomiji suivi du mot passé en paramètre, sans saut à la ligne.

```
void kaomijiOrateurln(String mot);
```

Cette procédure est l'équivalent du `println()` et affichera l'état de Kaomiji suivi du mot passé en paramètre, avec saut à la ligne.

- **Création, initialisation et affichage de la carte :**

```
void remplissageMap(String[][] map);
```

Cette procédure remplira le tableau à deux dimensions de type `String`, passé en paramètre, avec la chaîne de caractères "■" représentant la constante `CHEMIN`.

```
void elementMap  
(String[][] map, String[] tab, double probabilite);
```

Cette procédure ajoutera aléatoirement dans le tableau à deux dimensions de type `String` des éléments, comme la LAVE "🔥", les BOMBE "💣", etc., qui se trouvent dans un tableau à une dimension de type `String`, en fonction d'une `probabilité` d'être ajoutés.

```
void objectifMap(String[][] map, Objectif but);
```

Cette procédure placera aléatoirement, dans la moitié supérieure de la carte, un drapeau "🚩".

```
void placementJoueur(String[][] map, Joueur ludophile);
```

Cette procédure placera, dans le coin inférieur gauche de la carte, le personnage du joueur en fonction du choix de personnage effectué par le joueur.

```
void initialisationMap  
(String[][] map, Joueur ludophile, Objectif but);
```

Cette procédure initialisera la carte en utilisant les procédures vues précédemment : `remplissageMap()`, `elementMap()`, `objectifMap()` et `placementJoueur()`.

```
void afficherMap(String[][] map, Joueur ludophile);
```

Cette procédure affichera la carte, ainsi que les informations du joueur, telles que son nom, ses PV, ses coordonnées sur la carte et son nombre de réussites via `informationJoueur()`.

- **Vérification de la saisie du joueur :**

```
boolean estChiffre(String saisie);
```

Cette fonction retourne **False** si le caractère de la `saisie` n'est pas un chiffre compris entre '0' et '9' dans la table ASCII, sinon, par défaut, elle retournera **True**.

```
int stringToInt(String saisie);
```

Cette fonction convertit la `saisie` en un nombre (`int`), puis retourne ce nombre.

```
int verificationString(String saisie);
```

Cette fonction vérifie que la `saisie` du joueur est un chiffre en utilisant `estChiffre()`, puis retourne la conversion de cette `saisie` en nombre grâce à `stringToInt()`.

- **Vérification des déplacements du joueur :**

```
boolean deplacementPossibleNord  
(Joueur ludophile, String[][] map);
```

Cette fonction retourne **False** si le joueur a une MONTAGNE "▲" au-dessus de lui ou si son indice de ligne est égal à 0. Sinon, elle retourne **vrai** par défaut.

```
boolean deplacementPossibleSud  
(Joueur ludophile, String[][] map);
```

Cette fonction retourne **False** si le joueur a une MONTAGNE "▲" en-dessous de lui ou si son indice de ligne est égal à la taille du tableau - 1. Sinon, elle retourne **vrai** par défaut.

```
boolean deplacementPossibleOuest  
(Joueur ludophile, String[][] map);
```

Cette fonction retourne **false** si le joueur a une MONTAGNE " noen " à sa gauche ou si son indice de colonne est égal à 0. Sinon, elle retourne **vrai** par défaut.

```
boolean deplacementPossibleEst  
(Joueur ludophile, String[][] map);
```

Cette fonction retourne **false** si le joueur a une MONTAGNE " noen " à sa droite ou si son indice de colonne est égal à la taille du tableau - 1. Sinon, elle retourne **vrai** par défaut.

- **Déplacements du joueur :**

```
void avancerNord(Joueur ludophile, String[][] map);
```

Cette procédure fera, si le déplacement est possible, **monter** le joueur d'une case sur la carte.

```
void avancerSud(Joueur ludophile, String[][] map);
```

Cette procédure fera, si le déplacement est possible, **descendre** le joueur d'une case sur la carte.

```
void avancerEst(Joueur ludophile, String[][] map);
```

Cette procédure fera, si le déplacement est possible, **déplacer** le joueur d'une case vers la **droite** sur la carte.

```
void avancerOuest(Joueur ludophile, String[][] map);
```

Cette procédure fera, si le déplacement est possible, **déplacer** le joueur d'une case vers la **gauche** sur la carte.

```
void boucleCompteur(Joueur ludophile, String[][] map);
```

Cette procédure fera, si le déplacement est possible, **déplacer** le joueur de **n** cases dans la direction souhaitée.

```
void boucleWhile(Joueur ludophile, String[][] map);
```

Cette procédure fera, si le déplacement est possible et en fonction une condition, **déplacer** le joueur de **n** cases dans la direction souhaitée.

- **Création, vérification et exécution d'un algorithme :**

```
int[] creationAlgorithme();
```

Cette fonction retournera un tableau d'entiers de taille 20, représentant un déplacement (haut, bas, droite, gauche) ou un outil de déplacement (comme une boucle "pour" ou "while").

```
int[] confirmationAlgorithme  
(Joueur ludophile, String[][] map);
```

Cette fonction affiche l'algorithme choisi par le joueur à l'aide de la fonction précédente, afin que le joueur puisse confirmer ou non l'algorithme qu'il a composé. Ensuite, elle retourne un tableau d'entiers correspondant à la fonction vue précédemment.

```
void executionAlgorithme  
(Joueur ludophile, String[][] map, Objectif but);
```

Cette procédure parcourra le tableau d'entiers retourné par la fonction `confirmationAlgorithme()` pour effectuer des déplacements selon les valeurs du tableau. Si le joueur atteint le drapeau, il gagne. Sinon, il recommence.

- **Création et informations du personnage du joueur :**

```
String genreJoueur(Joueur ludophile);
```

Cette fonction retournera la saisie du joueur indiquant s'il est de genre masculin ou féminin, et servira à la sélection du personnage.

```
void afficherPersonnage(String[] personnage);
```

Cette procédure affichera les personnages présents dans le tableau qui sera passé en paramètre.

```
int selectionPersonnage(String[] personnage);
```

Cette fonction retournera un entier correspondant à l'emplacement d'un personnage dans le tableau passé en paramètre.

```
String personnageJoueur(Joueur ludophile);
```

Cette fonction reprend les fonctions précédentes afin de retourner le personnage que le joueur incarnera dans le jeu.

```
void recaputatif(Joueur ludophile);
```

Cette procédure récapitulera toutes les informations que le joueur aura saisies (nom, genre et personnage).

```
String positionJoueur  
(Joueur ludophile, String[][] map);
```

Cette fonction retournera la position du joueur sur la carte, permettant ainsi au joueur de se repérer et d'être utilisée pour d'autres fins, comme les déplacements ou les interactions.

```
String immuniteToString(Joueur ludophile);
```

Cette fonction retournera, selon l'état de `ludophile.immunite`, une chaîne de caractères indiquant si l'immunité est activée (`true`) ou désactivée (`false`).

```
String nbCoeur(int nbLife);
```

Cette fonction retournera une chaîne de caractères composée de "❤" en fonction du nombre de vies du joueur. Si le nombre de vies est égal à 0, elle retournera "💔".

```
String nbBouclier(Joueur ludophile);
```

Cette fonction retournera une chaîne de caractères composée de "🛡" en fonction du `ludophile.nbBouclier` du joueur. Si le nombre de bouclier est égal à 0, elle retournera 0.

```
void informationJoueur  
(Joueur ludophile, String[][] map);
```

Cette procédure récupère le nom, le nombre de vies, la position du joueur, le nombre de boucliers, l'immunité du joueur face aux malus, et les affiche principalement en dessous de la carte via la fonction `afficherMap()`.

- **Effets des éléments de la carte (bombe, montagne, etc.) :**

```
boolean objectifPasAtteint  
(Joueur ludophile, String[][] map, Objectif but);
```

Cette fonction retournera `true` si le drapeau rouge "🚩" est toujours à sa place sur la carte, sinon elle retournera `false`.

```
void effetBombe(Joueur ludophile, String[][] map);
```

Cette procédure vérifie s'il y a une bombe "💣" près du joueur et garde en mémoire où elle se trouve. Si le joueur va à l'endroit où se trouve la bombe, il explose "💥".

```
void effetArbre(Joueur ludophile, String[][] map);
```

Cette procédure vérifie s'il y a un arbre "🌴" près du joueur et garde en mémoire où elle se trouve. Si le joueur va à l'endroit où se trouve l'arbre, il a 50% de chance de se prendre une noix de coco sur la tête "🥥".

```
void effetLave(Joueur ludophile, String[][] map);
```

Cette procédure vérifie s'il y a de la lave "🌋" près du joueur et garde en mémoire où elle se trouve. Si le joueur va à l'endroit où se trouve la lave, il brûle "🔥".

```
void effetCarte(Joueur ludophile, String[][] map);
```

Cette procédure vérifie s'il y a une carte "🎴" près du joueur et garde en mémoire où elle se trouve. Si le joueur va à l'endroit où se trouve la carte, selon sa réponse, il obtiendra un **bonus** ou un **malus**.

```
void bonus(Joueur ludophile, String[][] map);
```

Cette procédure gère tous les bonus du jeu (récupération de vie, bouclier, immunité contre les malus), et c'est le joueur qui décide ce qu'il veut prendre.

```
void malus(Joueur ludophile, String[][] map);
```

Cette procédure gère tous les malus du jeu (astéroïdes, tornades), et c'est le destin qui décide du sort du joueur.

- **Fonctions liées au début du jeu :**

```
void begin(Joueur ludophile, Objectif but);
```

Cette procédure liste les procédures suivantes - `creationPersonnage()`, `tutoriel()` et `reglement()` - et c'est celle-ci qui lance le début du jeu.

```
void creationPersonnage(Joueur ludophile);
```

Cette procédure crée le personnage que le joueur va incarner en fonction de ce qu'il choisit (nom, genre, personnage).

```
void tutoriel(Joueur ludophile, Objectif but);
```

Cette procédure liste tous les éléments et procédures liés au tutoriel, surtout si le joueur est nouveau.

```
void reglement(Joueur ludophile, Objectif but);
```

Cette procédure rappelle les règles du jeu.

- **Fonctions liées au tutoriel :**

```
void avancerTutorial  
(Joueur ludophile, Objectif but, String[][] map);
```

Cette procédure initiera le nouveau joueur à l'outil du jeu '**avancer/haut**' à travers le **tutoriel()**.

```
void droiteTutorial  
(Joueur ludophile, Objectif but, String[][] map);
```

Cette procédure initiera le nouveau joueur à l'outil du jeu '**droite**' à travers le **tutoriel()**.

```
void gaucheTutorial  
(Joueur ludophile, Objectif but, String[][] map);
```

Cette procédure initiera le nouveau joueur à l'outil du jeu '**gauche**' à travers le **tutoriel()**.

```
void basTutorial  
(Joueur ludophile, Objectif but, String[][] map);
```

Cette procédure initiera le nouveau joueur à l'outil du jeu '**reculer/bas**' à travers le **tutoriel()**.

```
void boucleCompteurTutorial  
(Joueur ludophile, Objectif but, String[][] map);
```

Cette procédure initiera le nouveau joueur à l'outil du jeu '**boucle à compteur/pour**' à travers le **tutoriel()**.

```
void boucleWhileTutorial  
(Joueur ludophile, Objectif but, String[][] map);
```

Cette procédure initiera le nouveau joueur à l'outil du jeu '**boucle tant que/while**' à travers le **tutoriel()**.

```
void tutorielAlgorithme  
(Joueur ludophile, Objectif but, String[][] map);
```

Cette procédure initiera le nouveau joueur à l'**algorithme**, en utilisant les outils disponibles (haut, bas, etc.) à travers le **tutoriel()**.

- **Les conditions utilisées pour les boucles Tant Que :**

```
boolean estCheminNord  
(Joueur ludophile, String[][] map);
```

Cette fonction retournera **true** si un chemin "■" se trouve au-dessus du joueur, sinon elle retournera **false** s'il n'y a pas de chemin ou si le joueur est à la limite de la carte.

```
boolean estCheminSud(Joueur ludophile, String[][] map);
```

Cette fonction retournera **true** si un chemin "■" se trouve en-dessous du joueur, sinon elle retournera **false** s'il n'y a pas de chemin ou si le joueur est à la limite de la carte.

```
boolean estCheminEst(Joueur ludophile, String[][] map);
```

Cette fonction retournera **true** si un chemin "■" se trouve à droite du joueur, sinon elle retournera **false** s'il n'y a pas de chemin ou si le joueur est à la limite de la carte.

```
boolean estCheminOuest  
(Joueur ludophile, String[][] map);
```

Cette fonction retournera **true** si un chemin "■" se trouve à gauche du joueur, sinon elle retournera **false** s'il n'y a pas de chemin ou si le joueur est à la limite de la carte.

- **Choix de déplacements globaux et des boucles :**

```
void deplacement  
(int choix, Joueur ludophile, String[][] map);
```

Cette procédure déplacera le joueur en fonction de l'entier passé en paramètre, qui correspond à un type de déplacement.

```
void choixDeplacementBoucle  
(int nbChoix, int nbCases, Joueur ludophile, String[][]  
map);
```

Cette procédure, utilisée par `boucleCompteur()`, déplacera le joueur de manière répétée en fonction de l'entier passé en paramètre "nbChoix", qui indique le type de déplacement et le nombre de fois que le déplacement doit se répéter "nbCases".

```
void choixDeplacementWhile  
(int nbChoix, Joueur ludophile, String[][] map);
```

Cette procédure, utilisée par `boucleWhile()`, déplacera le joueur de manière répétée en fonction de l'entier passé en paramètre "nbChoix" et d'une condition choisie par le joueur.

- **Erreur lors de l'exécution de l'algorithme :**

```
void erreurAlgorithme  
(Joueur ludophile, String[][] map, Objectif but);
```

Cette procédure, utilisée par la plupart des fonctions de déplacement et en dehors du `tutoriel()`, fera recommencer le joueur à son emplacement initial si l'une des fonctions de déplacement lui indique que son déplacement n'est pas possible.