**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 (PVs)** : Le joueur commence avec un total de 10 PVs.
2. **Lave** : Si le joueur touche la lave "🔥", il perd 5 de ses PVs.
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 PVs** : Lorsque le joueur perd des PVs, Kaomiji s’énerve. Si les PVs du joueur atteignent 0, Kaomiji abattra le joueur.
7. **Cartes événements** : Des cartes événements "🎴" peuvent donner des bonus ou des malus au joueur.
8. **Statistiques** : À la fin du jeu, les statistiques du joueur seront envoyées dans un fichier séparé, lui permettant de comparer ses données avec celles des autres joueurs.

Une image contenant capture d’écran, texte, carte de visite, Police

Description générée automatiquement

**Description du code :**

* **Création de type :**

Joueur newJoueur();

Le type **Joueur**, c'est une personne qu'on crée pour commencer un jeu. On lui donne un nom (String), un genre (String), un personnage (String), un nombre de vies (int), un nombre de réussites (int) et tout ce qui concerne l'endroit où il va jouer, comme la ligne (int) et la colonne (int) où il se trouve, ainsi que sa position sur la carte (String).

Objectif newObjectif();

Le type **Objectif**, c'est le drapeau rouge "🚩" que le joueur doit atteindre pour gagner des réussites. 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 "🗻" 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 "🗻" 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.