2CPP : Documentation technique

- BERGUE William - MOUSSA OSMAN Elias -

```
#include <algorithm>
#include <iostream>
#include <map>
#include <vector>
using namespace std;
```

- `#include <algorithm>` : Cette ligne inclut la bibliothèque d'algorithmes standard de C++. Elle contient un ensemble de fonctions généralement utilisées pour diverses opérations telles que la recherche, le tri, la manipulation, etc.
- `#include <iostream>` : Cette ligne inclut la bibliothèque d'E/S standard qui fournit les fonctionnalités pour lire et écrire des données à partir de et vers les flux d'E/S standard (par exemple, le clavier et l'écran).
- `#include <map>` : Cette ligne inclut la bibliothèque map standard qui fournit une structure de données qui peut stocker des paires clé-valeur.
- `#include <vector>` : Cette ligne inclut la bibliothèque vector standard qui fournit une structure de données qui peut stocker une collection d'éléments (de même type) où l'ajout/suppression d'éléments peut être effectué à la fin.
- `using namespace std;` : Cette ligne est utilisée pour inclure l'espace de noms standard "std". Cela évite d'avoir à écrire "std::" devant chaque utilisation d'une fonction ou d'un objet qui appartient à cet espace de noms.

`struct Tiles` : Ceci déclare une structure de données appelée “Tiles”. Elle vas nous servir pour les tuiles, elle comprend :

- `int id` : Ceci est un membre de la structure “Tiles” qui est un entier appelé “id”. Il sert à stocker un identifiant unique pour chaque instance de “Tiles”.
- `int playerName` : Ceci est un autre membre de la structure “Tiles” qui est un entier appelé “playerName”. elle est utilisé pour stocker le numéro du joueur qui la pose.
- `vector<vector<int>> tile` : Ceci est un membre de la structure “Tiles” qui est un vecteur bidimensionnel d’entiers appelé “tile”. Il est utilisé pour stocker une grille de tuiles, où chaque tuile est représentée par un entier.

La classe `Player` définit les caractéristiques et les actions d'un joueur dans le jeu.

Attributs

- `string name`: Le nom du joueur.
- `int couponEchange`: Le nombre de coupons d'échange que possède le joueur.
- `int colorInt`: L'indice de la couleur choisie par le joueur.

Méthodes

`Player(vector<string> availableColors)` : Constructeur

- Initialise un nouveau joueur.
- Demande au joueur de saisir son nom en utilisant la méthode `choseName()`.
- Initialise le nombre de coupons d'échange à 1.
- Demande au joueur de choisir une couleur en utilisant la méthode `choseColor()`.

`vector<int> choseStartingPosition()`

- Demande au joueur de choisir une position de départ sur le plateau.

- Saisit le numéro de ligne et de colonne et les retourne sous forme de vecteur d'entiers.

`string choseName()`

- Demande au joueur de saisir son nom.
- Si le nom est "patate", le remplace par l'emoji 🥔.
- Retourne le nom saisi.

`int choseColor(vector<string> availableColors)`

- Affiche les couleurs disponibles avec leurs indices.
- Demande au joueur de choisir une couleur en saisissant l'indice.
- Vérifie si la couleur choisie est valide (non déjà prise et dans les limites).
- Retourne l'indice de la couleur choisie.

```cpp
class Game {
 public:
  vector<Player> players;
  vector<Tiles> tiles;
  vector<vector<int>> board;

  Game() {…

  void initBoard(vector<Player> players) {…

  vector<Tiles> initTilesStandard() {…

  void initTilesFromNowere() {…

  void printBoard() {…

  void printTile(Tiles tile) {…

  void placeTile(Player player) {…

  void exchangeTile() {…

  void printCurrentTile() {…

  bool canPlaceTile(Tiles tile, vector<int> position) {…

  void flipTile(Tiles tile) { reverse(tile.tile.begin(), tile.tile.end()); }

  void rotateTile(Tiles tile) {…

  void play() {…
};
```

La classe `Game` est conçue pour représenter et gérer l'état du jeux. Elle inclut des fonctionnalités pour initialiser le jeu, gérer les joueurs, les tuiles et le plateau de jeu, ainsi que des méthodes pour permettre aux joueurs de jouer leurs tours.

Constructeur

`Game()`

- Initialise une nouvelle partie.
- Invite l'utilisateur à saisir le nombre de joueurs (entre 2 et 9).
- Demande s'il faut utiliser des anciennes ou nouvelles tuiles et initialise les tuiles en conséquence.

- Initialise le plateau de jeu en fonction du nombre de joueurs.
- Pour chaque joueur, invite l'utilisateur à définir leur position de départ sur le plateau.

Méthodes publiques

`void initBoard(vector<Player> players)`

- Initialise le plateau de jeu en fonction du nombre de joueurs.
- Invite chaque joueur à définir sa position de départ sur le plateau.

`vector<Tiles> initTilesStandard()`

- Initialise l'ensemble standard de tuiles utilisées dans le jeu.
- Retourne un vecteur d'objets `Tiles` représentant les tuiles.

`void initTilesFromNowere()`

- Initialise un ensemble aléatoire de tuiles pour le jeu.
- Génère des tuiles avec des formes aléatoires et les ajoute à la liste des tuiles.

`void printBoard()`

- Affiche l'état actuel du plateau de jeu dans la console.
- Affiche le plateau avec les couleurs et symboles des joueurs.

`void printTile(Tiles tile)`

- Affiche la forme d'une tuile donnée dans la console.

`void placeTile(Player player)`

- Invite le joueur à placer une tuile sur le plateau.
- Vérifie si le placement est valide et met à jour le plateau en conséquence.

`void exchangeTile()`

- Permet au joueur d'échanger une tuile avec une tuile de la réserve.
- Affiche les cinq prochaines tuiles au joueur et lui demande un échange.

`void printCurrentTile()`

- Affiche la forme de la tuile actuelle détenue par le joueur.

`bool canPlaceTile(Tiles tile, vector<int> position)`

- Vérifie si une tuile donnée peut être placée à la position spécifiée sur le plateau.
- Vérifie si le placement est valide en fonction des règles du jeu.

`void flipTile(Tiles tile)`

- Retourne horizontalement la tuile donnée.

`void rotateTile(Tiles tile)`

- Tourne de 90 degrés dans le sens des aiguilles d'une montre la tuile donnée.

`void play()`

- Lance la boucle de jeu, permettant aux joueurs de jouer à tour de rôle.
- Affiche l'état actuel du plateau et de la tuile du joueur.
- Propose des options pour placer, échanger, faire pivoter et retourner des tuiles.
- Continue jusqu'à la fin de la partie.

Ce code est la fonction principale, il lance une nouvelle partie du jeu en créant une instance de la classe `Game` et en appelant la méthode `play()`.

```cpp
vector<Tiles> initTilesStandard() {
  vector<Tiles> tiles_ = {
      {1, 0, {{1, 0, 0}, {1, 1, 1}}},
      {2, 0, {{0, 1, 0}, {0, 1, 0}, {1, 1, 1}}},
      {3, 0, {{0, 1, 0}, {1, 1, 1}, {0, 1, 0}}},
      {4, 0, {{0, 0, 1}, {1, 1, 1}, {1, 0, 0}}},
      {5, 0, {{0, 1, 0}, {1, 1, 1}}},
      {6, 0, {{1, 1}, {1, 1}}},
      {7, 0, {{1, 0, 1}, {1, 1, 1}}},
      {8, 0, {{1, 1, 1}}},
      {9, 0, {{0, 1}, {1, 1}, {1, 0}}},
      {10, 0, {{1, 0}, {1, 1}}},
      {11, 0, {{0, 0, 1}, {0, 1, 1}, {1, 1, 0}}},
      {12, 0, {{1, 1}}},
      {13, 0, {{0, 1}, {1, 1}, {1, 0}, {1, 0}, {1, 1}}},
      {14, 0, {{1, 1, 1}, {1, 0, 0}, {1, 0, 0}, {1, 0, 0}, {1, 0, 0}}},
      {15,
       0,
       {{0, 0, 0, 1, 0},
        {0, 0, 0, 1, 0},
        {0, 0, 0, 1, 1},
        {0, 1, 1, 1, 0},
        {1, 1, 0, 0, 0}}},
  };

  return tiles_;
}
```

La fonction `initTilesStandard()` initialise un ensemble standard de tuiles pour le jeu. Chaque tuile est représentée par un objet de type `Tiles` avec un identifiant unique, un joueur associé, et une matrice représentant la forme de la tuile.