

Game Metro Mind Mapping

Viewer

Deck class

this class defines the deck of tiles. In this class we can determine whether the deck is empty, which helps us to determine whether the game is finished.

```
public int getNoOfTiles (Deck d)
public boolean isGameOver(int noOfTiles)
```

Board Class

Created the whole board 8x8
method: to place the tile
method: to test whether the position is empty or the placement is valid

Trails Class

This class defines each type of trail, namely a, b, c, d.

Tiles

enum Tiles

this enum type defines every type of tiles, and each with different trails on it.

Interface: TileInterface

- abstract method: to generate a map to store the information of the next connected tile based on the current tile, including the position and the exit index which track would start


```
nextTile(int exitIndex, int currentX, int currentY)
```
- method: to generate a map to record the maximum number of each tile


```
setTilesMap()
```
- method: to determine whether there is a tile whose count exceeds its maximum number


```
exceedMaxNumber(String placement)
```

Class: 24 tile classes (for each type of tile)

- Instance variable: its maximum method
- override the abstract method based on the tile itself

e.g. the current tile is aacb, the exit index is 0, the position is (y,x), then the next tile's position would be (y-1,x), the exit index is 0

split each tile from the given sequence and record its count, then compare with its maximum number

Station Class

Instance variable: 4 ArrayList to record the stations according to their direction (upper, below, left, right)

method: to generate a map based on the number of players, to match the plays and the stations belong to them

```
getPlayersStations(int numberOfPlayers)
```

method: to determine whether the track arrives at the stations or the centre stations

```
isArriveStation(int currentX, int currentY, int exitIndex)
isArriveCentreStation(int currentX, int currentY, int exitIndex)
```

be convenient to determine whether the tile is connected to station

to serve the get score function

Score Class

method: to calculate the points on one given station

```
getPoints(int stationPosition, String placementSequence)
```

Get the position and the exit index of a track that starts from the given station, then get the information of the next tile with the method nextTile(int exitIndex, int currentX, int currentY), repeat it until ArriveStation() is true. Add 1 point when the track cross a tile, finally check the total points and double it if isArriveCentreStation() is true.

Player Class

Instance variable: the number of the players

method: to get the points of each player

```
public int getPlayerX()
```

method: public Player getPlayer()

method: public Player generatePlayer(int noOfPlayer)

method: public Tiles getHandOfPlayer(Player player)

method: public boolean isHandEmpty(Player player)

method: public Tiles[] getPlayerTiles(Player player)

the player class defines players with their number and the stations they own. In this class we get to check what tiles each player has placed, and the tile each player draws from deck on his turn.

Functions for game process

To determine whether the tile is well-formed

```
isPiecePlacementWellFormed() // Task 2
```

To determine whether the placement sequence is well-formed

```
isPlacementSequenceWellFormed() // Task 3
```

To pick a tile from the deck randomly

```
drawFromDeck() // Task 5
```

Attention: do not exceed the maximum number of each tile

To determine whether the placement sequence is valid

```
isPlacementSequenceValid() // Task 6
```

Attention: each tile must be connected by the station or another tile, tiles cannot be overlapped, tiles cannot loop back to itself or another tile with a length of 1

To generate a valid placement

```
generateMove() // Task 9
```

Attention: the placement can only be in a valid position and the updated placement sequence should be valid

To get the final scores of each player

```
getScore() // Task 7
```

Attention: get station group with the method getPlayersStations in Statto class, get the points for each station with the method getPoints in Score method

More detailed information about Metro Class and Method

- **Metro Class**

This class will provide several methods to determine whether the tile and the current placement sequence are valid. It will also provide the method for players to place the tile and keep track of the placement sequence all the time. Finally, it will help compute the final points each player has got.

```
public class Metro {  
    /**  
     * Determine whether a tile is well-formed.  
     * 1. six characters for each tile (first four: "a-d", last two: digits  
between 0-7)  
     */  
    public static boolean isPiecePlacementWellFormed(String piecePlacement)  
{}  
  
    /**  
     * Determine whether a place sequence string is well-formed.  
     * 1. Be composed with well-formed tile  
     * 2. The number of each tile should be no more than its maximum number  
     */  
    public static boolean isPlacementSequenceWellFormed(String placement) {}  
  
    /**  
     * Draw a random tile from the deck.  
     * 1. Pick one tile from the tilesMap randomly  
     * 2. The number of the chosen tile must be more than 0 based on the  
tiles that have already been played  
     */  
    public static String drawFromDeck(String placementSequence) {}  
  
    /**  
     * Determine if a given placement sequence follows the rules of the game.  
     * 1. All tiles must have connection with another tile or the station  
     * 2. No piece can overlap another piece, or any of the central stations  
     *    ((3,3), (3,4), (4,3), (4,4))  
     * 3. The tile on the edge cannot loop back to itself
```

```

        * 4. The tile on the corner cannot have a track loop back to the
adjacent station

        * 5. No more than 60 tiles on the board

    */

    public static boolean isPlacementSequenceValid(String placementSequence)
    {}

    /**
     * Determine if a given placement sequence follows the rules of the game.
     * 1. Distinguishes which player the stations belong to
     * 2. Keeps track of points that the track from the starting station to
the end station
     *     (The tile's position touch the edge or the centre station)
     * 3. Computes the scores for each station and finally gets the total
points for each player
     */

    public static int[] getScore(String placementSequence, int
numberOfPlayers) {}

    /**
     * Given a placement sequence string, generate a valid next move.
     * 1. The position to put tile must be empty and choses the empty
position randomly
     * 3. The number of players determines the number of valid stations
     * 4. Check if it is a valid sequence string if the title is placed
     * 5. Choses to put the piece or hand randomly if both of them have valid
next move
     */

    public static String generateMove(String placementSequence, String piece,
String hand, int numberOfPlayers) {}
}

```

- **Tile Class**

There will be 26 classes for 26 different types of tiles. Each class will implement the Tile Interface which will provide several methods as follows:

```

public Interface TileInterface {
    public static HashMap<String, Integer> tilesMap = null;
}

```

```

/**
 * Generates a map for tiles, key is the tile's content and value is the
total number of that tile.
 * @return the tile map
 */
public static HashMap<String, Integer> setTilesMap() {}

/**
 * Gets the information of the next connected tile including the position
and the exit index that the track will start in the next connected tile.
 * @param exitIndex the exit index that the track starts
 * @param currentX the coordinate of x of the tile's current location
 * @param currentY the coordinate of y of the tile's current location
 * @return the map containing position as well as the exit index of the
next connected tile
 */
public abstract HashMap<String, Integer> nextTile(int exitIndex, int
currentX, int currentY);

/**
 * Check whether the number of each tile exceeds its total number.
 * @param placement A String representing the placement of all tiles on
the board
 * @return True if there is a tile's number exceeds its total number
 */
public static boolean exceedMaxNumber(String placement) {}

```

- **Station Class**

This class will have several methods to help give the different lists of stations depends on the number of players. Besides, it will also offer methods to check whether the tile connects to the station or the centre station.

```

public class Station {

    public static ArrayList<Integer> upperStations = new ArrayList<>();
    public static ArrayList<Integer> belowStations = new ArrayList<>();
    public static ArrayList<Integer> leftStations = new ArrayList<>();
    public static ArrayList<Integer> rightStations = new ArrayList<>();

```

```

public Station() {
    Collections.addAll(upperStations, 1, 2, 3, 4, 5, 6, 7, 8);
    Collections.addAll(belowStations, 17, 18, 19, 20, 21, 22, 23, 24);
    Collections.addAll(leftStations, 9, 10, 11, 12, 13, 14, 15, 15);
    Collections.addAll(rightStations, 25, 26, 27, 28, 29, 30, 31, 32);
}

/**
 * Generates different station map based on the number of the players,
the key is the player index, value is the stations that belong to him
 */
public static HashMap<Integer, ArrayList<Integer>> getPlayersStations(int
numberOfPlayers){}

/**
 * Checks whether the track on the current position connects the station.
 */
public static boolean isArriveStation(int currentX, int currentY, int
exitIndex) {

    /**
     * Checks whether the track on the current position connects the centre
     * station.
     */
    public static boolean isArriveCentreStation(int currentX, int currentY,
int exitIndex) {}
}

```

- **Score class**

This class will help compute the points of one tract.

```

public class Score {
    /**
     * Generates different station map based on the number of the players,
the key is the player index, value is the stations that belong to him.
     */
    public static int getPoints(int stationPosition, String
placementSequence) {}
}

```

- **Trails class**

This class defines each type of trail, namely a, b, c, d.

```
public class Trails {  
    char trails;  
    int entrance;  
    int exit;  
  
    private int getExit(Trails t, int entrance){  
        return 0;  
    }  
}
```

- **Deck class**

This class defines the deck of tiles. In this class we can determine whether the deck is empty, which helps us to determine whether the game is finished.

```
public class Deck {  
    Tiles[] deck;  
    int noOfTiles = 64;  
  
    Deck(Tiles[] deck, int noOfTiles){  
        this.deck = deck;  
        this.noOfTiles = noOfTiles;  
    }  
  
    public int getNoOfTiles (Deck d){  
        return 0;  
    }  
  
    public boolean isGameOver(int noOfTiles){  
        return true;  
    }  
}
```

- **Board class**

This class implements the appearance of the game in GUI, including the image of the board, clickable buttons in the window, and interactive information of the game. The

detailed arrangements are yet to be done.

```
public class Board {}
```

- **Player class**

The player class defines players with their number and the stations they own. In this class we get to check what tiles each player has placed, and the tile each player draws from deck on his turn.

```
public class Player {
    int PlayerNo;
    Station station;

    public Player(int playerNo, Station station){
        this.PlayerNo = playerNo;
        this.station = station;
    }

    public int getPlayerX(){
        return 0;
    }

    public Player getPlayer(){
        return null;
    }

    public Player generatePlayer(int noOfPlayer){
        return null;
    }

    public Tiles getHandOfPlayer(Player player){
        return null;
    }

    public boolean isHandEmpty(Player player){
        return true;
    }

    public Tiles[] getPlayerTiles(Player player){
```

```
        return null;  
    }  
}
```