Design Skeleton of Marrakech

Board

```
package comp1110.ass2;

public class Board {

    public final static int BOARD_WIDTH = 7;
    public final static int BOARD_HEIGHT = 7;

    private Square[][] boardMatrix;


    public Board(int rows, int columns) {
        boardMatrix = new Square[rows][columns];
        // initialize board Matrix
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                boardMatrix[i][j] = new Square();
            }
        }
    }

    private Rug[] rugs;

    Assam assam;

    private Square getSquareFromPos(IntPair position){
    }

    public boolean withinTrack(IntPair position) {

    }

    public String currentGame(){

    }




}
```

Player

```
package comp1110.ass2;

public class Player {
    ////So, for example, in the player string Pr00803i, we are representing the red player, who has 8 dirhams, 3 rugs remaining to place on

    public static char[] colors = {'c', 'y', 'r', 'p'};
    char playerColour; // there are 4 options for the colour of a player, cyan (represented by a c character), yellow (represented by a y c
    // cyan (represented by a c character), yellow (represented by a y character), red (represented by an r character), and purple (represe
    int dirhams; // playerStrings : This should always be a 3-digit number, so if a player has 6 dirhams that's represented as 006, and if
    int playerScore; // The player's score.
    int rugRemain = 15;// The number of Remaind Rug This is similarly always a 2-digit number
    Boolean inGame;
    Rug[] rugsForPlayer;

    public Rug[] getRugsForPlayer() {
        return rugsForPlayer;
    }

    public String getCurrentPlayer() {

    }

    String currentPlayer;
    public char getPlayerColour() {
        return playerColour;
    }

    public int getDirhams() {
        return dirhams;
    }

    public int getPlayerScore() {
        return playerScore;
    }

    public int getRugRemain() {
        return rugRemain;
    }

    public Boolean getInGame() {
        return inGame;
    }

    public Rug[] getRugs() {
        return rugs;
    }


    public void increaseDirhams(int amount) {

    }


    public void decreaseDirhams(int amount) {

    }


    public void decreaseDirhams(int amount) {

    }


}
```

Rug:

```
package comp1110.ass2;

public class Rug {
    private String color; // The color of the carpet tile.
    private int id; //the first rug placed be 00, then the next one be 01, then 02, etc is one sensible system
    private IntPair[] placementPosition;
    private boolean placed; //check if the rug is placed on the board

    public String toString(String color,int id,IntPair[] placementPosition){
    }

    public String toAbbreviatedString (String color,int id){
    }

    public String getColor() {
        return color;
    }

    public int getId() {
        return id;
    }

    public IntPair[] getPlacementPosition() {
        return placementPosition;
    }
    public boolean isPlaced() {

    }


}
```

Assam

```
package comp1110.ass2;

public class Assam {
    private IntPair currentLocation; // Rug where Assam is located
    private Facing assamFacing;
    private int moveSpace;

    private String currentAssam;

    private boolean isOnTrack;

    public IntPair getCurrentLocation() {
        return currentLocation;
    }

    public Facing getAssamFacing() {
        return assamFacing;
    }

    public int getMoveSpace() {
        return moveSpace;
    }

    public boolean isOnTrack() {
        return isOnTrack;
    }

    public String getCurrentAssam() {
        return currentAssam;
    }


    public void setCurrentLocation(IntPair currentLocation) {

    }

    public void setAssamFacing(Facing assamFacing) {

    }

    public void setMoveSpace(int moveSpace) {

    }

    public void setCurrentAssam(String currentAssam) {

    }

    public void setOnTrack(boolean onTrack) {

    }

    public void setSetCurrentLocation(void setCurrentLocation) {

    }

    public void setSetOnTrack(void setOnTrack) {

    }

    public void setSetAssamFacing(void setAssamFacing) {

    }
}
```

IntPair

```
package comp1110.ass2;

public class IntPair {
    private final int x;

    private final int y;

    public IntPair(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public IntPair add(IntPair other) {

        int newX = x + other.getX();
        int newY = y + other.getY();
        return new IntPair (newX,newY);

    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        IntPair position = (IntPair) o;
        return x == position.x && y == position.y;
    }


}
```

Square

```
package comp1110.ass2;

public class Square {
    private IntPair squarePosition; // absolute position of square on the board
    private Rug topRug = null; // the top rug on this Square
    private String topRugColour;
    private boolean assamStatus; // Check if Assam is on the square or not
    private boolean rugStatus; // Check if there is a rug on the square

    private Track track; // Associated track, if present
    private boolean hasTrack;
    public IntPair getSquarePosition() {

    }

    public Rug getTopRug() {

    }

    public boolean isAssamStatus() {

    }

    public boolean isRugStatus() {
    }

    public String getTopRugColour() {

    }

    public Track getTrack() {

    }

    public void setSquarePosition(IntPair squarePosition) {

    }

    public void setTopRug(Rug topRug) {

    }

    public void setTopRugColour(String topRugColour) {

    }

    public void setAssamStatus(boolean assamStatus) {

    }

    public void setRugStatus(boolean rugStatus) {

    }

    /**
     * Set the associated track for this square.
     * @param track The track to associate with this square.
     */
    public void setTrack(Track track) {

    }
    public boolean hasTrack() {

    }

    public void setHasTrack(boolean hasTrack) {

    }

}
```

Track

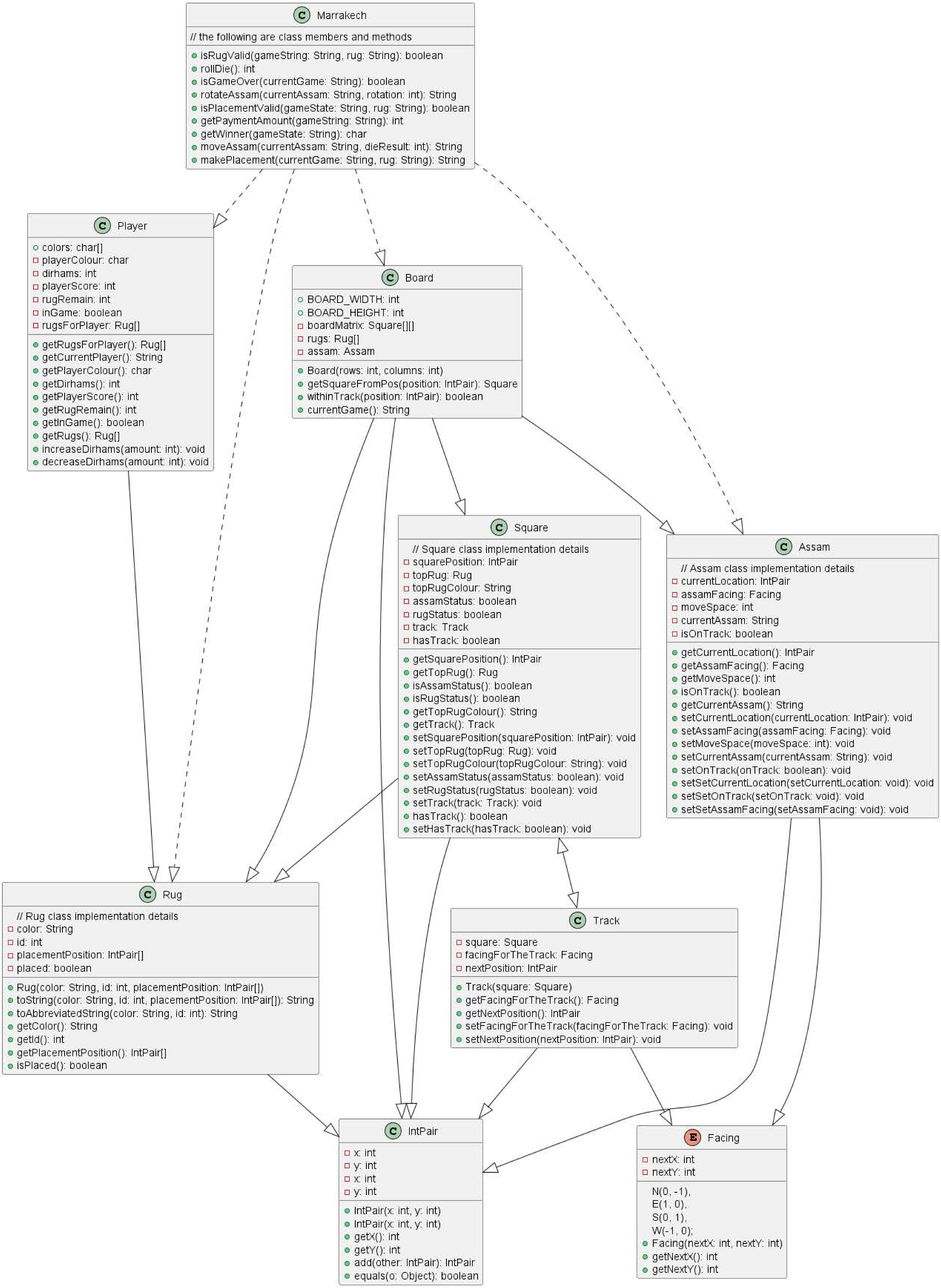
```
package comp1110.ass2;

public class Track {
    private Square square;
    private Facing facingForTheTrack;
    private IntPair nextPosition;

    public Track(Square square) {
        this.square = square;
    }

    public Facing getFacingForTheTrack() {
        return facingForTheTrack;
    }

    public IntPair getNextPosition() {
        return nextPosition;
    }

    public void setFacingForTheTrack(Facing facingForTheTrack) {
        this.facingForTheTrack = facingForTheTrack;
    }

    public void setNextPosition(IntPair nextPosition) {
        this.nextPosition = nextPosition;
    }
}
```

Facing

```
package comp1110.ass2;

public enum Facing {
        N(0, -1),
        E(1, 0),
        S(0, 1),
        W(-1, 0);

        private final int nextX;
        private final int nextY;

        private Facing(int nextX, int nextY) {
            this.nextX = nextX;
            this.nextY = nextY;
        }

        public int getNextX() {
            return nextX;
        }

        public int getNextY() {
            return nextY;
        }


    }
```

Marrakech

```
package comp1110.ass2;

public class Marrakech {

    /**
     * Determine whether a rug String is valid.
     * For this method, you need to determine whether the rug String is valid, but do not need to determine whether it
     * can be placed on the board (you will determine that in Task 10 ). A rug is valid if and only if all the following
     * conditions apply:
     *  - The String is 7 characters long
     *  - The first character in the String corresponds to the colour character of a player present in the game
     *  - The next two characters represent a 2-digit ID number
     *  - The next 4 characters represent coordinates that are on the board
     *  - The combination of that ID number and colour is unique
     * To clarify this last point, if a rug has the same ID as a rug on the board, but a different colour to that rug,
     * then it may still be valid. Obviously multiple rugs are allowed to have the same colour as well so long as they
     * do not share an ID. So, if we already have the rug c013343 on the board, then we can have the following rugs
     *  - c023343 (Shares the colour but not the ID)
     *  - y013343 (Shares the ID but not the colour)
     * But you cannot have c014445, because this has the same colour and ID as a rug on the board already.
     * @param gameString A String representing the current state of the game as per the README
     * @param rug A String representing the rug you are checking
     * @return true if the rug is valid, and false otherwise.
     */
    public static boolean isRugValid(String gameString, String rug) {
        // FIXME: Task 4
        return false;
    }

    /**
     * Roll the special Marrakech die and return the result.
     * Note that the die in Marrakech is not a regular 6-sided die, since there
     * are no faces that show 5 or 6, and instead 2 faces that show 2 and 3. That
     * is, of the 6 faces
     *  - One shows 1
     *  - Two show 2
     *  - Two show 3
     *  - One shows 4
     * As such, in order to get full marks for this task, you will need to implement
     * a die where the distribution of results from 1 to 4 is not even, with a 2 or 3
     * being twice as likely to be returned as a 1 or 4.
     * @return The result of the roll of the die meeting the criteria above
     */
    public static int rollDie() {
        // FIXME: Task 6
        return -1;
    }

    /**
     * Determine whether a game of Marrakech is over
     * Recall from the README that a game of Marrakech is over if a Player is about to enter the rotation phase of their
     * turn, but no longer has any rugs. Note that we do not encode in the game state String whose turn it is, so you
     * will have to think about how to use the information we do encode to determine whether a game is over or not.
     * @param currentGame A String representation of the current state of the game.
     * @return true if the game is over, or false otherwise.
     */
    public static boolean isGameOver(String currentGame) {
        // FIXME: Task 8
        return false;
    }

    /**
     * Implement Assam's rotation.
     * Recall that Assam may only be rotated left or right, or left alone -- he cannot be rotated a full 180 degrees.
     * For example, if he is currently facing North (towards the top of the board), then he could be rotated to face
     * East or West, but not South. Assam can also only be rotated in 90 degree increments.
     * If the requested rotation is illegal, you should return Assam's current state unchanged.
     * @param currentAssam A String representing Assam's current state
     * @param rotation The requested rotation, in degrees. This degree reading is relative to the direction Assam
     *                 is currently facing, so a value of 0 for this argument will keep Assam facing in his
     *                 current orientation, 90 would be turning him to the right, etc.
     * @return A String representing Assam's state after the rotation, or the input currentAssam if the requested
     * rotation is illegal.
     */
    public static String rotateAssam(String currentAssam, int rotation) {
        // FIXME: Task 9
        return "";
    }

    /**
     * Determine whether a potential new placement is valid (i.e that it describes a legal way to place a rug).
     * There are a number of rules which apply to potential new placements, which are detailed in the README but to
     * reiterate here:
     *   1. A new rug must have one edge adjacent to Assam (not counting diagonals)
     *   2. A new rug must not completely cover another rug. It is legal to partially cover an already placed rug, but
     *      the new rug must not cover the entirety of another rug that's already on the board.
     * @param gameState A game string representing the current state of the game
     * @param rug A rug string representing the candidate rug which you must check the validity of.
     * @return true if the placement is valid, and false otherwise.
     */
    public static boolean isPlacementValid(String gameState, String rug) {
        // FIXME: Task 10
        return false;
    }

    /**
     * Determine the amount of payment required should another player land on a square.
     * For this method, you may assume that Assam has just landed on the square he is currently placed on, and that
     * the player who last moved Assam is not the player who owns the rug landed on (if there is a rug on his current
     * square). Recall that the payment owed to the owner of the rug is equal to the number of connected squares showing
     * on the board that are of that colour. Similarly to the placement rules, two squares are only connected if they
     * share an entire edge -- diagonals do not count.
     * @param gameString A String representation of the current state of the game.
     * @return The amount of payment due, as an integer.
     */
    public static int getPaymentAmount(String gameString) {
        // FIXME: Task 11
        return -1;
    }

    /**
     * Determine the winner of a game of Marrakech.
     * For this task, you will be provided with a game state string and have to return a char representing the colour
     * of the winner of the game. So for example if the cyan player is the winner, then you return 'c', if the red
     * player is the winner return 'r', etc...
     * If the game is not yet over, then you should return 'n'.
     * If the game is over, but is a tie, then you should return 't'.
     * Recall that a player's total score is the sum of their number of dirhams and the number of squares showing on the
     * board that are of their colour, and that a player who is out of the game cannot win. If multiple players have the
     * same total score, the player with the largest number of dirhams wins. If multiple players have the same total
     * score and number of dirhams, then the game is a tie.
     * @param gameState A String representation of the current state of the game
     * @return A char representing the winner of the game as described above.
     */
    public static char getWinner(String gameState) {
        // FIXME: Task 12
        return '\0';
    }

    /**
     * Implement Assam's movement.
     * Assam moves a number of squares equal to the die result, provided to you by the argument dieResult. Assam moves
     * in the direction he is currently facing. If part of Assam's movement results in him leaving the board, he moves
     * according to the tracks diagrammed in the assignment README, which should be studied carefully before attempting
     * this task. For this task, you are not required to do any checking that the die result is sensible, nor whether
     * the current Assam string is sensible either -- you may assume that both of these are valid.
     * @param currentAssam A string representation of Assam's current state.
     * @param dieResult The result of the die, which determines the number of squares Assam will move.
     * @return A String representing Assam's state after the movement.
     */
    public static String moveAssam(String currentAssam, int dieResult){
        // FIXME: Task 13
        return "";
    }

    /**
     * Place a rug on the board
     * This method can be assumed to be called after Assam has been rotated and moved, i.e in the placement phase of
     * a turn. A rug may only be placed if it meets the conditions listed in the isPlacementValid task. If the rug
     * placement is valid, then you should return a new game string representing the board after the placement has
     * been completed. If the placement is invalid, then you should return the existing game unchanged.
     * @param currentGame A String representation of the current state of the game.
     * @param rug A String representation of the rug that is to be placed.
     * @return A new game string representing the game following the successful placement of this rug if it is valid,
     * or the input currentGame unchanged otherwise.
     */
    public static String makePlacement(String currentGame, String rug) {
        // FIXME: Task 14
        return "";
    }

}
```



```
C Marrakech
// the following are class members and methods
● isRugValid(gameString: String, rug: String): boolean
● rollDie(): int
● isGameOver(currentGame: String): boolean
● rotateAssam(currentAssam: String, rotation: int): String
● isPlacementValid(gameState: String, rug: String): boolean
● getPaymentAmount(gameString: String): int
● getWinner(gameState: String): char
● moveAssam(currentAssam: String, dieResult: int): String
● makePlacement(currentGame: String, rug: String): String
```

```
C Player
○ colors: char[]
□ playerColour: char
□ dirhams: int
□ playerScore: int
□ rugRemain: int
□ inGame: boolean
□ rugsForPlayer: Rug[]
● getRugsForPlayer(): Rug[]
● getCurrentPlayer(): String
● getPlayerColour(): char
● getDirhams(): int
● getPlayerScore(): int
● getRugRemain(): int
● getInGame(): boolean
● getRugs(): Rug[]
● increaseDirhams(amount: int): void
● decreaseDirhams(amount: int): void
```

```
C Board
○ BOARD_WIDTH: int
○ BOARD_HEIGHT: int
□ boardMatrix: Square[][]
□ rugs: Rug[]
□ assam: Assam
● Board(rows: int, columns: int)
● getSquareFromPos(position: IntPair): Square
● withinTrack(position: IntPair): boolean
● currentGame(): String
```

```
C Square
// Square class implementation details
□ squarePosition: IntPair
□ topRug: Rug
□ topRugColour: String
□ assamStatus: boolean
□ rugStatus: boolean
□ track: Track
□ hasTrack: boolean
● getSquarePosition(): IntPair
● getTopRug(): Rug
● isAssamStatus(): boolean
● isRugStatus(): boolean
● getTopRugColour(): String
● getTrack(): Track
● setSquarePosition(squarePosition: IntPair): void
● setTopRug(topRug: Rug): void
● setTopRugColour(topRugColour: String): void
● setAssamStatus(assamStatus: boolean): void
● setRugStatus(rugStatus: boolean): void
● setTrack(track: Track): void
● hasTrack(): boolean
● setHasTrack(hasTrack: boolean): void
```

```
C Assam
// Assam class implementation details
□ currentLocation: IntPair
□ assamFacing: Facing
□ moveSpace: int
□ currentAssam: String
□ isOnTrack: boolean
● getCurrentLocation(): IntPair
● getAssamFacing(): Facing
● getMoveSpace(): int
● isOnTrack(): boolean
● getCurrentAssam(): String
● setCurrentLocation(currentLocation: IntPair): void
● setAssamFacing(assamFacing: Facing): void
● setMoveSpace(moveSpace: int): void
● setCurrentAssam(currentAssam: String): void
● setOnTrack(onTrack: boolean): void
● setSetCurrentLocation(setCurrentLocation: void): void
● setSetOnTrack(setOnTrack: void): void
● setSetAssamFacing(setAssamFacing: void): void
```

```
C Rug
// Rug class implementation details
□ color: String
□ id: int
□ placementPosition: IntPair[]
□ placed: boolean
● Rug(color: String, id: int, placementPosition: IntPair[])
● toString(color: String, id: int, placementPosition: IntPair[]): String
● toAbbreviatedString(color: String, id: int): String
● getColor(): String
● getId(): int
● getPlacementPosition(): IntPair[]
● isPlaced(): boolean
```

```
C Track
□ square: Square
□ facingForTheTrack: Facing
□ nextPosition: IntPair
● Track(square: Square)
● getFacingForTheTrack(): Facing
● getNextPosition(): IntPair
● setFacingForTheTrack(facingForTheTrack: Facing): void
● setNextPosition(nextPosition: IntPair): void
```

```
C IntPair
□ x: int
□ y: int
□ x: int
□ y: int
● IntPair(x: int, y: int)
● IntPair(x: int, y: int)
● getX(): int
● getY(): int
● add(other: IntPair): IntPair
● equals(o: Object): boolean
```

```
E Facing
□ nextX: int
□ nextY: int
N(0, -1),
E(1, 0),
S(0, 1),
W(-1, 0);
● Facing(nextX: int, nextY: int)
● getNextX(): int
● getNextY(): int
```