MazeNode
<<Interface>>

+ setNode(MazeNode cell, Direction dir): void
+ getNode(int row, int col): MazeNode
+ getNode(Direction dir): MazeNode
+ canReach(MazeNode cell): boolean
+ wealthiestPath(MazeNode cell): Path
+ explore(): Path
+ directionTo(MazeNode cell): Direction

Edge
<<Interface>>

+ getTail(): MazeNode
+ getHead(): MazeNode

Path
<<Interface>>

+ reachesTarget(): boolean
+ totalGold(): int
+ getNodes(): List<MazeNode>

Wall

- tail: MazeNode
- head: MazeNode

MazePath

+ MazePath(Cell target)

CellNode

- north: Cell
- south: Cell
- east: Cell
- west: Cell
- gold: int

+ Cell(int row, int col)

BlockedNode

MazeBuilder
<<Interface>>

+ init(): MazeBuilder
+ setStart(int row, int col): MazeBuilder
+ setGoal(int row, int col): MazeBuilder
+ addPlayer(string name): MazeBuilder
+ createPaths(): MazeBuilder
+ build(): Maze

Maze
<<Interface>>

+ getPlayer(): Player
+ getCurrentCell(): Cell
+ move(Direction dir): Maze
+ explore(): Path
+ wealthiestPath(): Path

Player
<<Interface>>

+ getGoldCollected()
+ lootCell(Cell cell)

Player

+ Player(String name)

RoomMazeBuilder

- rowCount: int
- colCount: int
- remainingWallCount: int
- isWrapping: boolean
- cells: Cell[][]
- remainingWalls: List<Edge>
- percentGold: double
- percentThief: double
- thiefPenalty: double

+ StandardMazeBuilder(
    int rowCount,
    int columnCount,
    int remainingWallCount,
    double percentGold,
    double percentThief,
    double thiefPenalty,
    boolean isWrapping)

PerfectMazeBuilder

- rowCount: int
- colCount: int
- remainingWallCount: int
- isWrapping: boolean
- cells: Cell[][]
- remainingWalls: List<Edge>
- percentGold: double
- percentThief: double
- thiefPenalty: double

+ StandardMazeBuilder(
    int rowCount,
    int columnCount,
    double percentGold,
    double percentThief,
    double thiefPenalty,
    boolean isWrapping)

2DMaze

- start: Cell
- player: Player
- currentCell: Cell

+ Maze(
    Player player,
    Cell start)

Coordinate
<<Interface>>

+ getX(): int
+ getY(): int

MazeCoordinate

+ MazeCoordinate(x, y);

Direction
<<Enum>>

NORTH
EAST
SOUTH
WEST