Notes about Maze 1.MoveEnum MoveEnum a.stores four directions. 2.Each maze will have a 'wrapping' status. It could - MOVE_NORTH: MoveEnum = (-1, 0) be set during initializing. - MOVE_SOUTH: MoveEnum = (1, 0) 3.movePlayer(MoveEnum) - MOVE_EAST: MoveEnum = (0, 1) a.move player with a direction - MOVE_WEST = (0, -1) 4.getPlayer() a.return current player. So we could know the + MoveEnum(int, int) information of this player(location, gold, etc.) + getX(): int 5.setStartPosition(int, int) & setGoalPosition(int, int) + getY(): int a.we could reset start and goal positions. So we could reuse a maze. 6.showBestRoute() a.print best route for current player to get most gold MazeFactory <<interface>> Maze 7.randomStartAndGoal() a.randomly generate start and goal position. **Notes about Player.class** b.funnier 1.goldCount 8.store walls by Map<Cell, Set<Cell>> a.using long to store to avoid overflow a.O(1) look up time + getPossibleDirections(): List<MoveEnum> + generatePerfectMaze(int, int, boolean): Maze Count 2.move b. No pointer issue + generateRoomMaze(int, int, int, boolean): Maze + setPlayer(Player): void a.move player 'MoveEnum' steps 9.MazeFactory + getPlayer(): Player - mazeCount: Long = 0 3.pickGold + movePlayer(MoveEnum): void a.used to generate maze - playerCount: Long = 0 a.goldCount will increase by xxx + setStartPosition(int, int): void b.has two static methods 4.loseGold + setGoalPosition(int, int): void i. generatePerfectMaze(row, col, isWrapping) + showBestRoute(): void a.goldCount will decrease by xxx + createMaze(): Long ii. generateRoomMaze(row, col, wallCount, + randomStartAndGoal(): void 5.loseGoldByPercentage + createPlayer(): Long isWrapping) + getNumberOfMazes(): Long a.player will lose xxx% gold. 10. generateWalls(numOfWalls) + getNumberOfPlayers(): Long b.1 <= argument <= 100 a.generate 'numOfWalls' walls *AbstractMaze* b. could be used by both perfect maze and room maze since the only difference between them Location - mazeld: long is the number of walls - type: MazeTypeEnum - x: int - player: Player - row: int - y: int - col: int - wrapping: boolean **Test Plan for Maze** + Location(int, int): void - cells: Cell[][] + getX(): int - walls: Map<Cell, Set<Cell>> 1. Test correctly - wallCount: int + getY(): int 2.IllegalArgumentException Player - startPosition: Location + setX(int): void a.constructor - goalPosition: Location +setY(int): void negative row and col - playerId: long +setLocation(int, int): void b.movePlayer - name: String + AbstractMaze(MazeTypeEnum, int, int, wrapping) - location: Location i. null as argument + generateWalls(int) - goldCount: long = 0 c.setStartPosition i. out of bounds index ii. same position as goal position + Player(String) d.setGoalPosition + getPlayerId(): long i. out of bounds index + getName(): String + getLocation(): Location ii. same position as start position PerfectMaze RoomMaze + setLocation(Location): void e.generateWalls + move(MoveEnum): void i. number exceeds maximum + getGoldCount(): long f. generateRoomMaze + setGoldCount(long): void i. numOfWalls should be larger than or equal + pickGold(long): void + PerfectMaze(int, int, boolean) + PerfectMaze(int, int, boolean) to 0, while smaller than or equal to + loseGold(long): void numOfEdges - n + 1 + loseGoldByPercentage(int): void 3.OutOfBoundsException a.maze is non-wrapping and step out of the maze 4.IllegalStateException a. movePlayer i. player is null ii. Try to step toward a wall iii. No start position or goal position MazeTypeEnum PERFECT ROOM **Notes about Cell** 1.each Cell has a type -- CellTypeEnum 2.processPlayer(Player) a.process should be done by Cell. We'd better not expose properties of cells to maze. 3.GoldCell <<interface>> cell a.GoldCell will have a 'gold' parameter b.two constructors, need to declare gold in constructors. 4. ThiefCell and NormalCell don't have variables. The only difference between them is type. + isThiefCell(): boolean _____ 5.isAdjacent(Cell, boolean) + isGoldCell(): boolean a. This method will be used when we're + getLocation(): Location building walls. + setLocation(Location): void b.second argument will be 'isWrapping' + processPlayer(Player): void + isAdjacent(Cell, boolean): boolean **Test Plan for Cell** 1. Test correctly 2. Illegal Argument Exception a.constructor i. negative row and col AbstractCell CellTypeEnum ii. null location iii. null type - location: Location - NORMAL b.processPlayer - type: CellTypeEnum - GOLD i. null as argument - THIEF c. isAdjacent(cell, null) + AbstractCell(type, int, int) i. null as argument + AbstractCell(type, Location) GoldCell ThiefCell NormalCell - gold: long = 0+ Cell(int, int, long) + Cell(int, int) + Cell(int, int) + Cell(Location, long) + Cell(Location) + Cell(Location)

Test Plan for Player.class

1. Test correctly

a.constructor

b.setLocation

d.setGoldCount

e.pickGold

f. loseGold

a.loseGold

c. move

2. Illegal Argument Exception

i. null as argument

i. null as argument

i. null as argument

i. negative argument

i. negative argument

i. negative argumentg.loseGoldByPercentage

3. IllegalStateException

i. negative argument or zero

ii. argument larger than 100

i. argument larger than goldCount