HW3: UML: <<Interface>> <<Interface>> Maze Player + getTotalRow(): int + getCurrentGold(): int + getTotalCol(): int + pickUpCoins(): Coin + getWallNum(): int + loseCoins(): Thief Cell + getAllEdges(): List Δ + getCell(): cell[][] location: int[] + addCoins(): void thief: Thief Player + getCoinsLocation(): list coinValue: Coin + addThief(): void cellID: int currentCoins: int + getThiefLocation():list visited: boolean + buildMaze(): void directions: List public Player(){} northCell: Cell southCell: Cell + getCurrentGold(): int westCell: Cell + pickUpCoins(): Coin Δ eastCell: Cell + loseCoins(): Thief public Cell(int[][] location){} AbstractMaze + getLocation(): int [][] Game - COIN_PROBABILITY: double + setID: void THIEF_PROBABILITY: double + getID: int player: Player - totalRow: int + setThief(): void map: Cell[][] + getThief(): Thief - totalCol: int playLocation: int[] + setCoins(int coin): void - WallNum: int endPoint: int[] allEages: List + getCoins(int coin); Coin done: boolean goldLocationList: list + remove Coins(): void - thiefLocationList: list + setNorth(): void public Game(int[] startPoint + setSouth(): void - cell: cell[][] int[] endPoint + setWest(); void Cell[][] map){} public AbstractMaze(int col, + setEast(): void + getPlayLocation(): int[] int row. + getNorthCell(): Cell + getCurrentCoins(): int int remainWalls){ } + getSouthCell(): Cell + getOptions(): List + getTotalRow(): int + getWestCell(): Cell + move(): void + getTotalCol(): int + getEastCell(): Cell + update(): void + getWallNum(): int + getDirections: List<String> + done(): boolean + getAllEdges(): List + toString(): String + getCell(): cell[][] + addCoins(): void + getCoinsLocation(): list + addThief(): void + getThiefLocation():list Extends -Extends + buildMaze(): void Coin coinValue: int Extends public Coin(int coins){} + valueChange(): int WrappingRoomMaze RoomMaze PerfectMaze

public RoomMaze(int col,

int row,

int remainWalls){ }

Thief

COIN_LOSS_RATIO: double

loss: double

public Thief(){}

+ valueChange(): int

public WrappingRoomMaze(

int col.

int row.

int remainWalls){ }

Test Case:

1. Test Maze

public PerfectMaze(int col,

int row){ }

//Check maze Construction: toString()

//Check maze information: assertEquals(1, maze.getNumOfRows()) assertEquals(1, maze.getNumOfCols()) assertEquals(0, maze.getRemainWalls()) assertEquals(false, maze.isWrapped())

```
//test negative numOfRows/numOfCol: throw IllegalArgumentException
   Perfect maze = new PerfectMaze(1, -2);
   //test invalid remain walls
   PerfectMaze maze = new PerfectMaze(3,2,3) //should be between 0-2
   //test negative numOfRows/numOfCol
   assertEquals(1, maze.getNumOfRows())
   assertEquals(1, maze.getNumOfCols())
   //Check gold information:
   assertEquals(2, maze.getGoldLocation().get[0](2))
   //Check thief information:
   assertEquals(2, maze.getThiefLocation().get[0](2))
2. Test Player:
   //test current gold
   assertEquals(0, player.getCurrentGold());
   // testCollectGold
   // testLoseGold
   //start point/ goal point have negative num
   Play.setStartingPoint(0,-2);
3. Test Game:
   //testPlayLocation. After each move
     assertEquals(0, MazeGame.getPlayerLocation()[1]);
     MazeGame.move(SOUTH);
     assertEquals(1, newGame.getPlayerLocation()[0]);
   // testCurrentGold, after moving
```