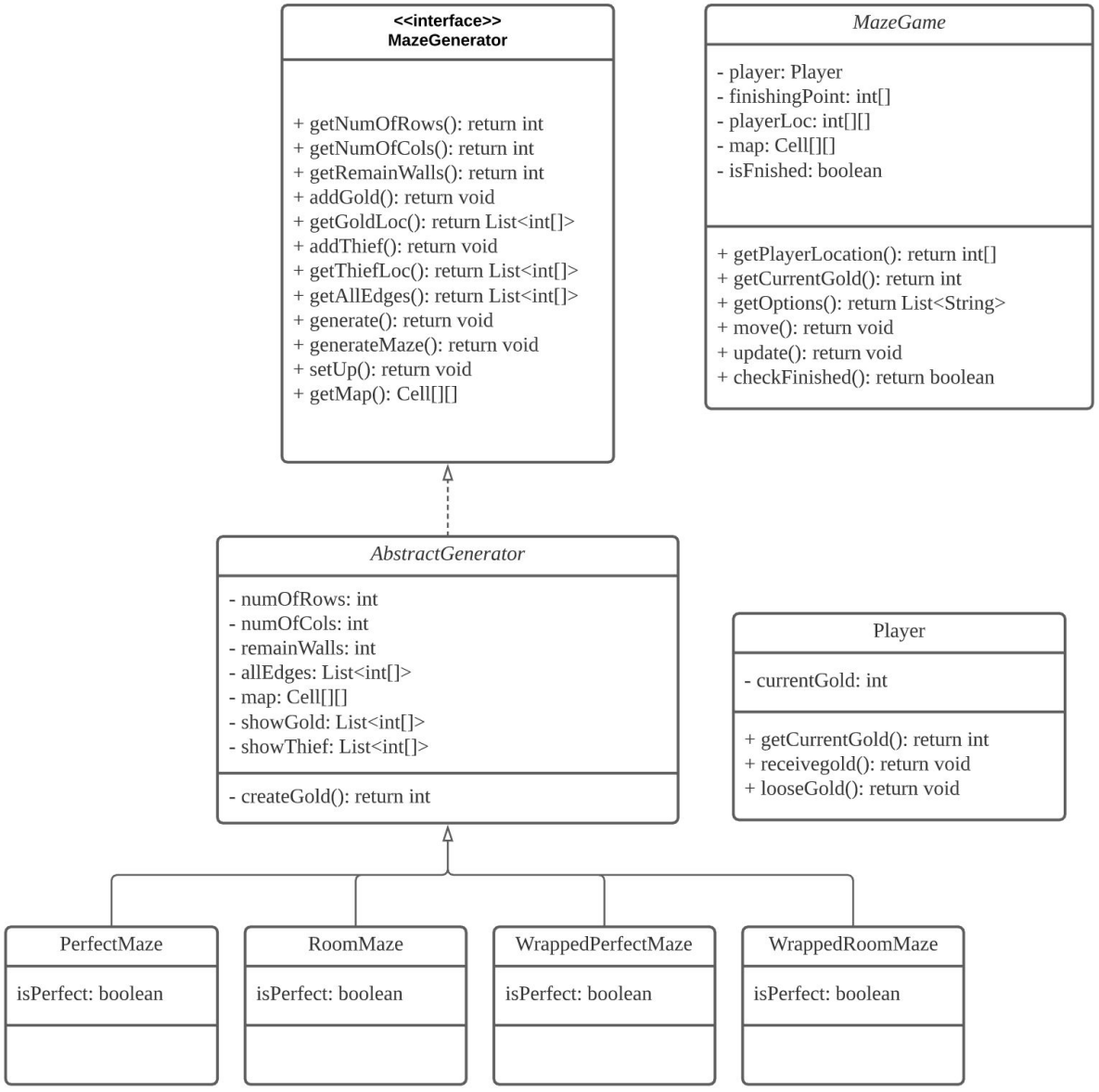
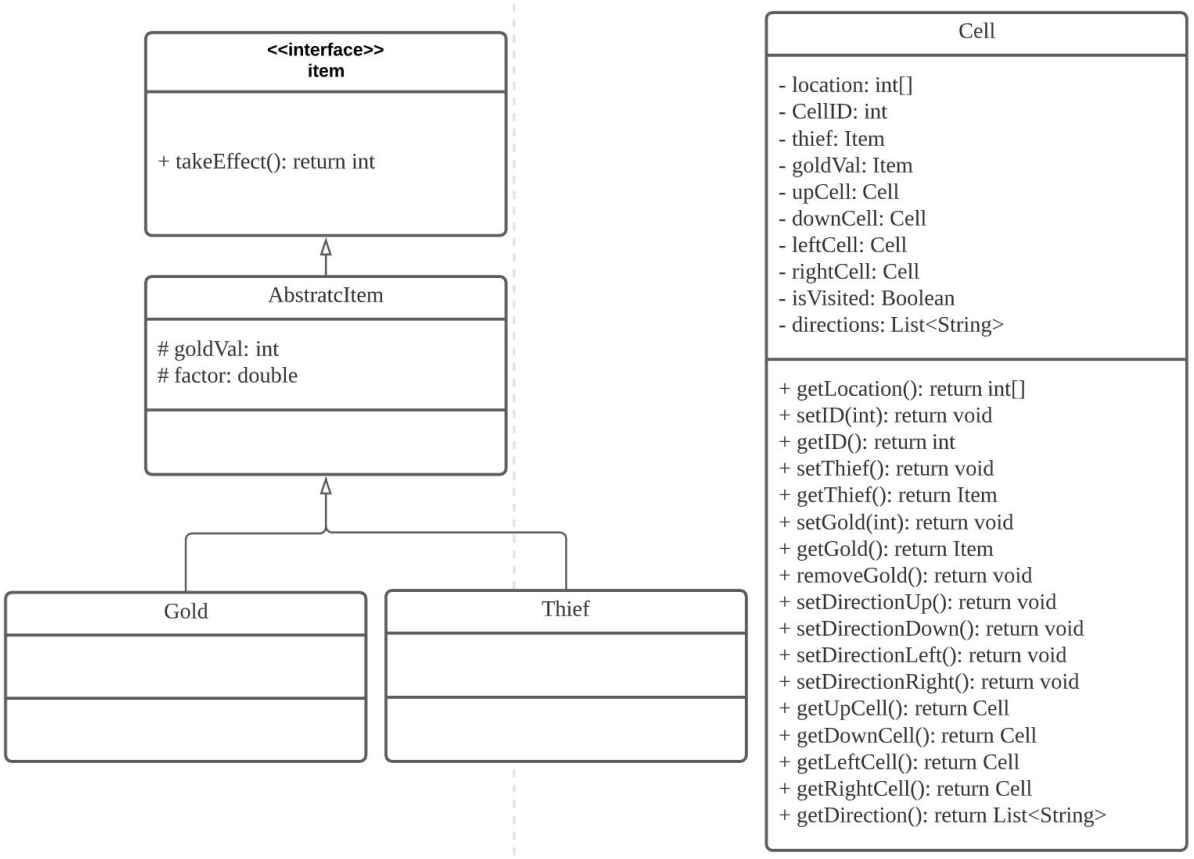
- UML diagram


<<interface>>
MazeGenerator
+ getNumOfRows(): return int
+ getNumOfCols(): return int
+ getRemainWalls(): return int
+ addGold(): return void
+ getGoldLoc(): return List<int[]>
+ addThief(): return void
+ getThiefLoc(): return List<int[]>
+ getAllEdges(): return List<int[]>
+ generate(): return void
+ generateMaze(): return void
+ setUp(): return void
+ getMap(): Cell[][]
MazeGame
- player: Player
- finishingPoint: int[]
- playerLoc: int[][]
- map: Cell[][]
- isFnished: boolean
+ getPlayerLocation(): return int[]
+ getCurrentGold(): return int
+ getOptions(): return List<String>
+ move(): return void
+ update(): return void
+ checkFinished(): return boolean
AbstractGenerator
- numOfRows: int
- numOfCols: int
- remainWalls: int
- allEdges: List<int[]>
- map: Cell[][]
- showGold: List<int[]>
- showThief: List<int[]>
- createGold(): return int
Player
- currentGold: int
+ getCurrentGold(): return int
+ receivegold(): return void
+ looseGold(): return void
PerfectMaze
isPerfect: boolean
RoomMaze
isPerfect: boolean
WrappedPerfectMaze
isPerfect: boolean
WrappedRoomMaze
isPerfect: boolean



- Test Case:
  1. Initializing Driver:
     Scan input: row, col, remainWall….

  2. Creating Maze:
     // Generate new maze according to the conditions
     Generator maze1 = new PerfectMaze(row, col, remainWalls)
     maze.generate()

     Generator maze2 = new WrappedRoomMaze(row, col, remainWalls)
     maze.generate()

  3. Build Game:
     MazeGame newGame = new MazeGame(startingPoint, finishingPoint, maze.getMap());

  4. Game Time:
     newGame.move(directions)
     newGame.checkFinished()

  5. Test:
     // check maze info
     assertEquals(4, maze1.getNumOfRows())
     assertEquals(6, maze1.getNumOfCols())
     assertEquals(0, maze1.getRemainWalls())

```
// check gold location
assertEquals(2, maze1.getGoldLoc().get(0)[0]);
assertEquals(0, maze1.getGoldLoc().get(0)[1]);
assertEquals(3, maze1.getGoldLoc().get(1)[0]);
assertEquals(1, maze1.getGoldLoc().get(1)[1]);

// test invalid remain walls
Generator obj3 = new PerfectMaze(5, 3, 9);  // 0~8


// check player location and move
assertEquals(0, newGame.getPlayerLocation()[0]);
assertEquals(0, newGame.getPlayerLocation()[1]);
newGame.move("down");
assertEquals(1, newGame.getPlayerLocation()[0]);
assertEquals(0, newGame.getPlayerLocation()[1]);
newGame.move("right");
assertEquals(1, newGame.getPlayerLocation()[0]);
assertEquals(1, newGame.getPlayerLocation()[1]);

// check player's direction options
List<String> test = new ArrayList<>();
test.add("down");
assertEquals(test, newGame.getOptions());

// check getting gold
newGame.move("down");
assertEquals(4, newGame.getCurrentGold());

// check finish
assertFalse(newGame.checkFinished());
finishingPoint = new int[]{1, 0};
maze = new PerfectMaze(4, 6, 0);
maze.generate();
newGame = new MazeGame(startingPoint, finishingPoint, maze.getMap());
newGame.move("down");
assertTrue(newGame.checkFinished());
```