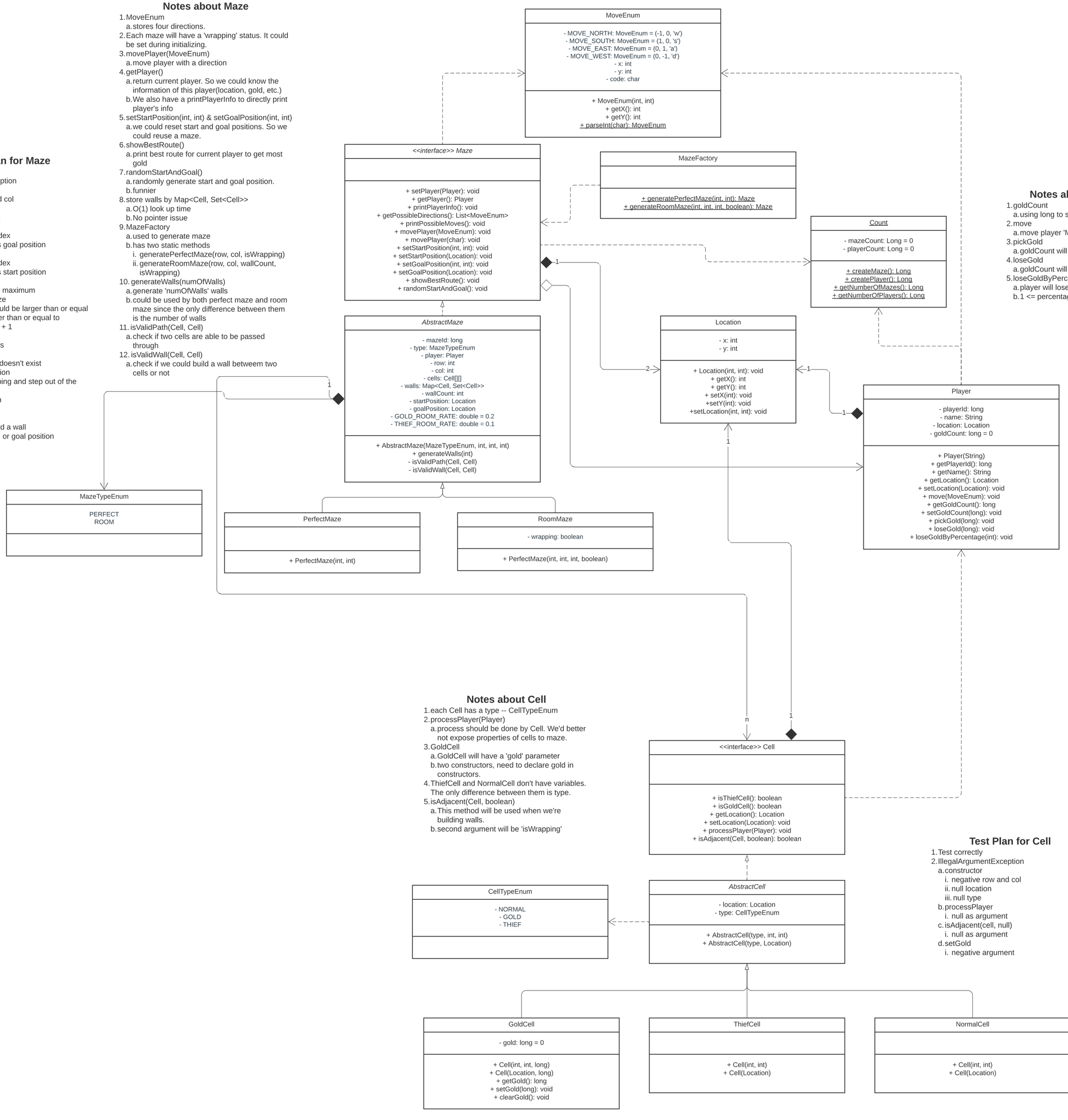
Notes about Maze

1. MoveEnum
   a. stores four directions.
2. Each maze will have a 'wrapping' status. It could be set during initializing.
3. movePlayer(MoveEnum)
   a. move player with a direction
4. getPlayer()
   a. return current player. So we could know the information of this player(location, gold, etc.)
   b. We also have a printPlayerInfo to directly print player's info
5. setStartPosition(int, int) & setGoalPosition(int, 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
7. randomStartAndGoal()
   a. randomly generate start and goal position.
   b. funnier
8. store walls by Map<Cell, Set<Cell>>
   a. O(1) look up time
   b. No pointer issue
9. MazeFactory
   a. used to generate maze
   b. has two static methods
      i. generatePerfectMaze(row, col, isWrapping)
      ii. generateRoomMaze(row, col, wallCount, isWrapping)
10. generateWalls(numOfWalls)
    a. generate 'numOfWalls' walls
    b. could be used by both perfect maze and room maze since the only difference between them is the number of walls
11. isValidPath(Cell, Cell)
    a. check if two cells are able to be passed through
12. isValidWall(Cell, Cell)
    a. check if we could build a wall betweem two cells or not

Test Plan for Maze

1. Test correctly
2. IllegalArgumentException
   a. constructor
      i. negative row and col
   b. movePlayer
      i. null as argument
   c. setStartPosition
      i. out of bounds index
      ii. same position as goal position
   d. setGoalPosition
      i. out of bounds index
      ii. same position as start position
   e. generateWalls
      i. number exceeds maximum
   f. generateRoomMaze
      i. numOfWalls should be larger than or equal to 0, while smaller than or equal to numOfEdges - n + 1
   g. isValidPath
      i. null as arguments
   h. movePlayer(int)
      i. instruction code doesn't exist
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



Notes about Player.class

1. goldCount
   a. using long to store to avoid overflow
2. move
   a. move player 'MoveEnum' steps
3. pickGold
   a. goldCount will increase by xxx
4. loseGold
   a. goldCount will decrease by xxx
5. loseGoldByPercentage
   a. player will lose xxx% gold.
   b. 1 <= percentage <= 100

Test Plan for Player.class

1. Test correctly
2. IllegalArgumentException
   a. constructor
      i. null as argument
   b. setLocation
      i. null as argument
   c. move
      i. null as argument
   d. setGoldCount
      i. negative argument
   e. pickGold
      i. negative argument
   f. loseGold
      i. negative argument
   g. loseGoldByPercentage
      i. negative argument or zero
      ii. argument larger than 100
3. IllegalStateException
   a. loseGold
      i. argument larger than goldCount

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
   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.
5. isAdjacent(Cell, boolean)
   a. This method will be used when we're building walls.
   b. second argument will be 'isWrapping'

Test Plan for Cell

1. Test correctly
2. IllegalArgumentException
   a. constructor
      i. negative row and col
      ii. null location
      iii. null type
   b. processPlayer
      i. null as argument
   c. isAdjacent(cell, null)
      i. null as argument
   d. setGold
      i. negative argument