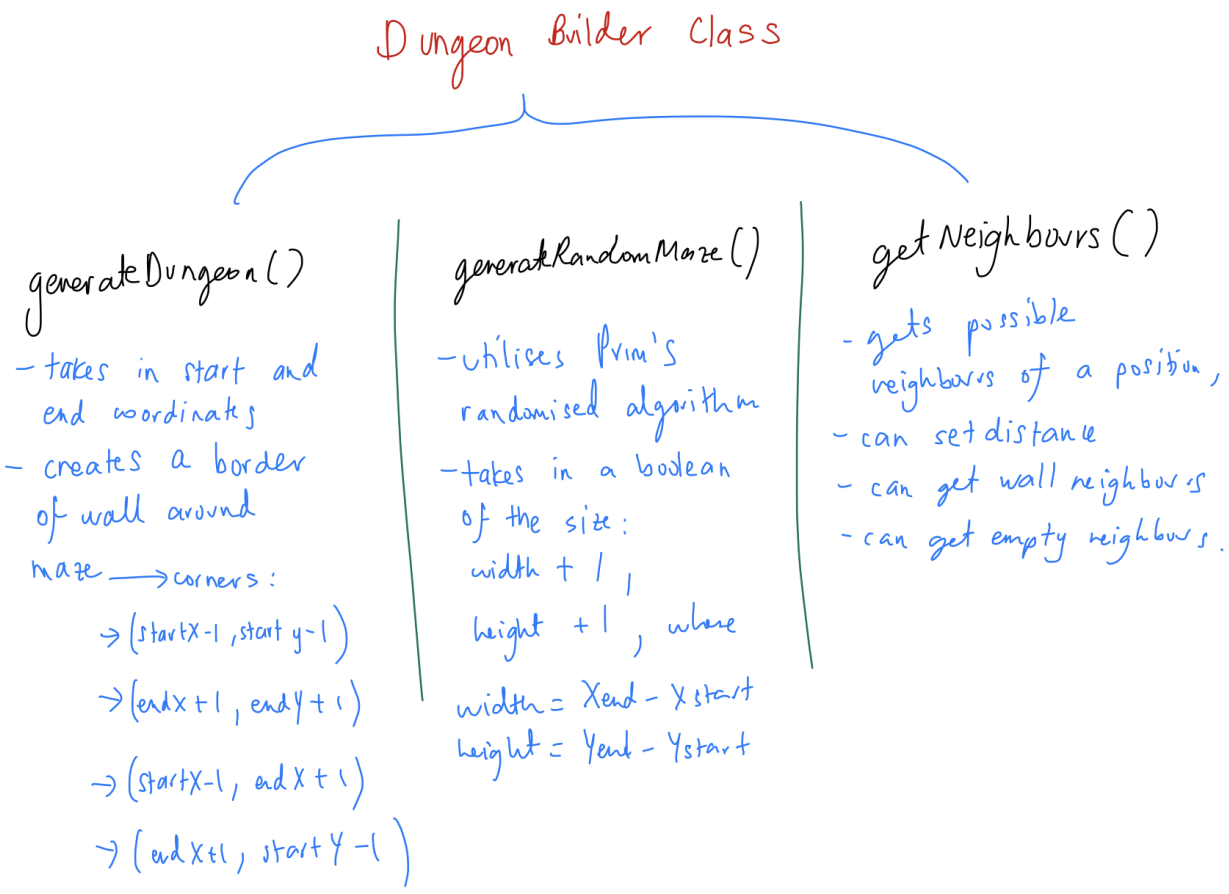
1. *Analyse the ticket requirements, including the technical and product specifications.*

- Dungeon builder will be build in the Dungeon.java class
- It will take in a start coordinate and end coordinate
- It will we bordered by walls
- It will utilise Prim's randomized algorithm to build the maze

2. *Using your high level design (UML diagram) from Milestone 1, on a piece of paper sketch out a detailed design for the ticket. This should include:*

    a. *What fields/methods you will need to add/change in a class*
    b. *What new classes/packages you will need to create*



3. *Have someone else in your team review the Detailed Design before you start work. One approach is to have, for each ticket, an Assignee (the person who works on the task) and a Reporter (someone who doesn't do the actual work but understands what's going on, and acts as a reviewer).*

Assignee: Ridho Yusuf
Reporter: Oudom Lim
Comments: I think the design so far is good but I think there are interfaces which we do not use often or not at all so we can probably remove that and use a new strategy.

4. *Go back and iterate on the design if needed.*

The dungeon builder will be placed in its own class, in order to move functionality out of the Dungeon class

5. *Once the design is approved, write a test list (a list of all the tests you will write) for the ticket. Map out each of the conceptual cases you want to test. This can just be written in a document, or if you want make function stubs for JUnit tests and put up a Merge Request.*

Testing plan:
    1. Check that config file does not exist
    2. Create Dungeon 30 by 30
    3. Create Dungeon of size 10 and 12
    4. Create Dungeon of size 2 and 2

6. *Have someone else in your team review the test list to make sure the test cases all make sense and cover the input space.*

Assignee: Ridho Yusuf
Reporter: David Peng
Comments: Moving dungeon builder out of the Dungeon class is a good idea as it reduces the complexity of the Dungeon class code.  I personally think that the testing plan is good as it checks each parameter individually so we know that it works as expected but it is also integrated with one another for example in bomb logic where you would use a switch and a wire to activate it, hence giving us smaller integration tests.

7. *Stub out anything you need to with class/method prototypes.*
8. *Write the tests, which should be failing assertions currently since the functionality hasn't been implemented.*

Tests written and made a merge request.

9. *Implement the functionality so that your tests pass.*
10. *Run a usability test (check your functionality works on the frontend).*
11. *Where needed, refactor your code to improve the style and design while keeping the tests passing.*

12. *Put up a merge request with your changes into master. The CI should be passing. The merge request should have a meaningful title and contain a description of the changes you have made. In most cases you should just be able to link them to your design/test list documents.*
13. *At least one other person in your team need to review and approve your merge request before you merge your branch into master.*
14. *Update your collective UML diagram with your detailed design.*

Updated.