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

    - The bomb entity will require one function, the detonate function.
    - The bomb entity will have one attribute, hasBeenPickedUp, which will ensure that a bomb that has been placed cannot be picked up again.
    - The bomb will destroy all entities (except the player) within its square bomb radius, including enemies if it is placed cardinally adjacent to an active switch.

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*

Bomb.java

ensures that a bomb that has been placed cannot be picked up again

Attributes:
- hasBeenPickedUp: boolean

Methods:
- detonate(): void
  ↳ detonate the bomb if:
    - the bomb is cardinally adjacent to an active switch
      ↳ must check if switch and floor switch are at the same position.
  ↳ destroy entities within bomb radius
    - must determine if a entity is within the bomb's radius
    - does not destroy the player

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: The detailed design covers all aspects of the functionality and carefully considers possible cases such as ensuring that player is not destroyed. Must consider what happens when the bomb is placed before the switch is activated.

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

    The design does consider what happens when the bomb is placed before the switch is activated because the detonate() function will be conducted at every tick and during every tick, it will check if any bombs need exploding. Therefore, this design will cover this certain case.

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. Test surrounding entities are removed when placing a bomb next to an active switch with config file bomb radius set to 2
   2. Test surrounding entities are removed when placing a bomb next to an inactive switch with config file bomb radius set to 2
   3. Try to pick up bomb that has already been placed

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: Rupin Bhattal
   Comments: Test was well written and incorporated what the group was referring to, it also includes a broad number of test cases.

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.*

    Tick is implemented in the bomb function, in order to reduce complexity. Tests are still 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.