*Space Panther*

Rongcui Dong (rd2848) and Olessya Medvedeva (oam2113)

**Expand each user story into a *use case*. Each use case should include at least title, actor(s), description, basic flow and alternate flows (when applicable). Focus on the basic flow vs. alternate flows, which should specify the step by step actions of the user (or external system using your product) and what your product does in response.**

Use case 1

**Title**: Have a single planet simulated in the game

**Actor**: user of the game

**Description**: The user would like to simulate movements of a planet with certain mass, start coordinates and velocity

**Basic Flow**:

1. The user clicks on the game editor’s run button
2. The game editor opens up a game with the main\_scene screen that has a start menu with 3 buttons that allow user to select form: play a new game, load the previous game or exit
3. The user clicks on play the game button
   1. Or exit button
4. The game moves to the main menu screen that displays the information that the user needs to enter in order to start the game: mass, coordinates and velocity. The message box prompts the user to enter numeric values that are floats (with precision of 4 symbols after the dot). In addition, main menu has 3 buttons: add planet, play and exit.
   1. The software closes the game screen and the user cannot play anymore.
5. The user enters the data which could be one of the following:
   1. The user enters positive float/integer values for mass, positive or negative float/integer values for each of the coordinates (x, y, z) and velocities (Vx, Vy, Vz).
   2. The user enters negative float/integer values for mass, positive or negative float/integer values for each of the coordinates and velocities.
   3. The user enters non numeric values for all parameters.
   4. The user enters non numeric value for mass and the rest are positive or negative numeric values.
   5. The user enters positive numeric value for mass and a non numeric value for one of the other parameters or all of the rest of parameters.
   6. Or presses the play button, then the flow goes to 8.a.i
6. The response of the software is triggered after user enters the information in the text cell for:
   1. No error messages or warning show up
   2. The message is displayed on top to enter only positive numeric values for mass since it cannot be negative
   3. The messages to enter numeric values are displayed right after the user enters non numeric value in the cell and for each of the cell that gets the non numeric value
   4. The message is displayed to enter numeric value for mass
   5. As 6c but no message for the mass comes out
7. The user can choose:
   1. Press Add planet button
      1. Or press Play button
   2. Do 7a.
      1. or follow the displayed message and re-enter value for the mass which could follow any of the 6a-e
   3. to e are the same as 7b.
8. The response of the software:
   1. After pressing the add planet button, the software checks one more time that the values for mass are numeric and positive and the rest of the parameters are numeric positive or negative values. Since data will pass the check - it will be saved in to the globally available singleton’s class dictionary. And the software will do 8.a.i.1.
      1. Or if pressed play button, then the software checks if the data was saved:
         1. If it was, then the menu scene is deleted and the software moves to the play scene to simulate the planet
         2. If it was not saved, then the message is displayed to save the planets info before pressing the play button on the same screen
   2. to c if did 7 a, then will follow 8 a if pressed add button, but will not pass the data check since it was not corrected, so software will redisplay the same helpful messages, or will follow 8.a.i.2 and then 8a if pressed play button
      1. If the user corrected the entered data, the software will remove the error messages
9. The user :
   1. Can press exit button which will prompt the software to exit the game and display the start menu
   2. After correcting the input, can press the one of the buttons, which will follow the 8a

10. The user can press exit button, then the software will exit, or choose to play another game, which will start the flow from 2.

Use case 2

**Title**: Have multiple planets simulated in the game

**Actor**: user of the game

**Description**: The user would like to simulate movements of several planets with certain mass, start coordinates and velocity

**Basic Flow**:

1. will be the same upto 8, then after entering the data, the user can:
   1. Press add button:
      1. If the input is valid (as was discussed above) the software will add the data to the planets\_dict
         1. The user then can:
            1. Press the play button:

If only 1 planet was entered , then the flow will follow the flow of a single planet game as discussed earlier used case

If more than one planet was entered than planets will be added to the simulation in order they were entered by the user

* + - * 1. Enter another set of data, in this case the flow will follow the same as discussed above till 8. and will start this case’s flow from 1.

Then the flow will repeat from 1

* + - * 1. Exit the game, then the software will display the start menu
  1. Press play button:
     1. The software will check if the current data was saved:
        1. If it was saved:
           1. Then the flow will go from 1.a.i.1a.
        2. If was not saved, the user will be prompted to save the data and the flow will start from 1
  2. Press exit button
     1. The software will move to the start menu with the choice of playing a new game or exiting the game for good.

**For your test plan, explain the equivalence partitions and boundary conditions necessary to unit-test each of the major subroutines in your system (functions, procedures, methods, etc. excluding getters/setters and helpers), with references to your specific test case(s) addressing each equivalence partition and each boundary condition. Your test suite should include test cases from both valid and invalid equivalence partitions, and just below, at, and just above each boundary condition. State where your test suite resides in your github repository, but you do not need to copy the test code into this document.**

The input validation tests mostly focus on sanitizing floating point inputs to be used into physics simulation. Therefore, the equivalence classes are mostly physically significant. The code reside in *tests/test-nbodySimulator.cpp*, which constructs test cases using the Doctest framework.

Mass validation has equivalence classes of zero, negative mass, positive mass, positive and negative infinite mass, and NaN. All of these classes are tested. In addition, the boundary cases of +DBL\_EPSILON, -DBL\_EPSILON, +DBL\_MAX, and -DBL\_MAX are tested.

Similarly, velocity validation tests include all classes and boundary conditions. Since velocity is a 3-vector, each class appears in x, y, z at least once.

Since coordinate is also a 3-vector, the same equivalence classes and boundaries are used.

Then, the simulator is tested on a solar system with known solution. The position vector is tested against a golden result.

Unit tests for the godot script classes are included in tests/menu and includes tests of the methods for the main classes used in the menu GUI: “Global.gd”, “main\_grid.gd”, “main\_menu\_scene.gd”. No tool at the moment available to measure the coverage for the code in GDScript that was used to write these classes.

**Then describe how you measured the branch coverage achieved by your test suite.**

*Gcov* is used to measure test coverage. A gcov report is generated after tests are run and printed on Travis log.

Make sure to include a link to the github repository where your entire codebase resides.

<https://github.com/wixyFun/openSpace>