|  |  |  |
| --- | --- | --- |
| Student Name | CSE Account | Handin Account |
|  |  | [ ]  |
|  |  | [ ]  |
|  |  | [ ]  |
|  |  | [ ]  |
|  |  | [ ]  |

|  |
| --- |
| **Program Features and Functionality (30% plus stars for bonus options)** |
| Interactively plots an animated 3-D lunar equatorial orbit, allowing user control of impulse thrusts to modify orbit (30%)  |  |
| Bonus – Gives choice of moons/planets/comets/etc. (with appropriate parameters) |  |
| Bonus – Renders more realistically the orbited objects |  |
| Bonus – Enables non-equatorial orbits |  |
| Bonus – Shows rotation of the body being orbited |  |
| Bonus – Implements a version of “Lunar Lander” game, enabling bringing the satellite out of orbit and (hopefully) to a soft landing |  |
| Bonus – Has calculator mode so user can run relevant orbital formulas before entering new thrust parameters |  |
| Bonus – Add a second orbiting satellite and implement a game involving docking with it |  |
| **Comments:** |
| **Program Logic (20% plus a star for bonus)** |
| Efficient coding and design. (5%) |  |
| Modular organization with testable modules. (5%) |  |
| Minimal code redundancies. (5%) |  |
| Iterative development of code. (5%) |  |
| Bonus – sharing of a module, without any alternations, with another team (with appropriate attribution, of course!) |  |
| **Comments:** |
| **Programming Style (20%)** |
| Headers. (5%) |  |
| Descriptive variable names. (5%) |  |
| Spacing and indentation. (5%) |  |
| Judicious commenting. (5%) |  |
| **Comments:** |
| **Documentation (30%)** |
| Cover page with name(s) and the account under which it is submitted, title, date submitted, etc. (5%) |  |
| A discussion of the features you implemented in the project. High-level description of how they work and what Matlab options were used to program them. (5%)  |  |
| Acknowledgement of all collaborations (both internal to the team and external), detailing what each person contributed individually, and what was done jointly. Indicate approximate percentages of the work contributed by each person in design, coding, testing, documentation, and report preparation. (5%) |  |
| Discussion of testing, including how each module was tested; printouts showing tests may be helpful (10%) |  |
| User manual. (5%) |  |
| Note: All documentation plus a copy of the Matlab code must be neatly and clearly packaged in one document; (Code must also be submitted electronically separately so that graders can test it.) |  |
| **Comments:** |
| **Total Points (100%)** |  |
| **Additional Notes:** |

Here we give some examples and guidelines on what we expect of each component. Please, read it carefully, and consider these guidelines while working on your homeworks and projects.

**Program Functionality**

This refers to:

1. whether and how well the program meets the requirements specified in the assignment,
2. how well the program addresses the problem specified in the assignment, and

**Program Logic and Organization**

This refers whether:

1. the overall software organization is clear and sensible,
2. the program file naming convention and data file naming convention.
3. whether and how logically correct the solution is. It is possible that your code may yield incorrect results with logically correct solution to the problem specified in the assignment. It is also possible that your code may yield correct results with logically incorrect solution to the problem specified in the assignment.

A solution that gives correct results may be poorly designed. Software clarity or simplicity, consistency, modularity, extensibility, and maintainability are a few important principles of good software design.

**Programming Style**

This refers to:

1. good and clear indentations (e.g., a tab or at least three spaces for an indentation),
2. good use of variable names (e.g., numberOfStudents is better than x),
3. proper declaration of global constants,
4. efficient use of functions to prevent redundant coding,
5. adequate length of each method functions, and so on.

You should see a lot of good examples of good programming style from our textbook. The consistency of your programming style throughout your code is also extremely important. It allows readers of your code to understand your code much more easily. Our textbook provides a lot of pointers on good programming styles.

**Testing**

This refers to our testing of your program, using our own input data file and matching your output to the correct output. Boundary conditions will be tested to see whether your program can handle those conditions. For example, suppose that your program reads an input integer from the screen and processes it. Will your program still function if instead a real number is entered?

**Documentation**

This refers to all the good practices that one should practice: cover page, header documentation (program), inline documentation and README file. (Note: For our class projects the README will be part of the “User Manual” section.) Since we will cover this in our laboratory, I will not elaborate it here. But a good documentation should include:

1. A proper cover,
2. description of what the code does,
3. description of how the code achieves what it wants to do
4. at times, why the code uses a particular solution strategy, and
5. a README file is a description of all the files used: including the source
file(s), input files (if any), output files (if any). It should also include an
instruction on how to run your program. The purpose of a README file is to let your readers know how to use your programs and data files.