Design & Professional Skills
Assignment 2: Debugging the *Bomber* Game

Instructions

In the `benning-teaching/ENGF34-2024` github repository there are three Python programs in the `assignments/assignment2` directory. These are three implementations of exactly the same very simple very dumb *Bomber* game. This game is a re-implementation of a game written for the Sinclair ZX81 which originally ran in 1 KByte of RAM.

The objective of the game is to level the city by dropping bombs on the buildings so you can land the plane. The plane gets lower each time it crosses the screen. If you fly into a building, you die. If you succeed in flattening all the buildings, you can land the plane and score 1000 points. Your reward is to start all over again, but with narrower buildings that are harder to hit.

The three versions of the game should behave identically, but are written in different programming styles:

- `bomber_bigloop.py` uses a lot of global variables and one big main loop.
- `bomber_proc.py` uses a procedural style, with fewer global variables and breaks the code into many functions. It makes extensive use of python lists to pass data into and out of these functions.
- `bomber_oo.py` uses an object-oriented style, defining classes for the plane, buildings, etc, and functions that operate on those classes.

The game requires python 3 and tkinter.

The problem is that the games have **at least five bugs that make them unplayable**. Your task is:

1. Play the game.
2. Find a bug.
3. Write a brief bug report describing the bug.
4. Identify the cause of the bug. Write a brief summary of the cause.
5. Fix the bug.
6. Repeat from 1.

Bug Reports

A bug report should be brief and to the point. It should include:

- One sentence summary of the bug.
- Description of what happens.
- Description of what you think should happen.
- Instructions for how to reproduce the bug.

Understanding the bug

Once you've written the bug report, look at the code. You can look at all three versions, or just one version — whatever you find easiest.

Identify what the code is doing when the bug is triggered. Sometimes the cause may be obvious from reading the code. Often the cause is not obvious, and even the flow of the code may not be obvious. Then you will need to instrument the code to figure out what it is doing. In this case, I just want you to instrument the code using `print()` - there's no need to use a debugger. Generally, you want to instrument the code without changing its behaviour until you gather enough information to understand what the code is actually doing (and hence why it differs from what it should be doing).

Another common debugging technique is to reduce the code complexity by removing code to reduce it to the simplest case that still exhibits the buggy behaviour. This is a valuable technique when a bug is hard to reproduce.

Another technique is to make things more deterministic. For example, in this game, the buildings are randomly generated heights. You might call `random.seed(x)` with a constant parameter `x`, so the buildings are always generated the same way. By manually choosing different values of `x` you can settle on one that makes it simpler to generate the buggy behaviour in a repeatable manner. Of course you'd then need to remove the fixed seed later, when you've fixed the bug.

In general, you're hunting for evidence until you've found out what the program is actually doing. Only when you understand what the program is doing should you think about how to fix it.

Fixing the bugs

These bugs are very simple. Some are one line fixes, none requires more than about three lines of extra code to fix. **You only need to fix the bugs in one of the three versions of the code**, but you can choose which version you work with.

Assessment

This assignment is not assessed. The purpose of the assignment is to give you practice reading, understanding, and debugging a non-trivial piece of code. You many work with your friends on this assignment if you wish. You must hand in a reasonable attempt via Moodle to receive a binary mark, as we want to see how you are progressing. Hand in a zipfile containing the text of your bug reports, you brief explanations of the causes of the bugs, and the python source code of **one** of the three versions of the game with the bugs fixed.

Optional Extra

A few members of the class are more experienced programmers. If you find fixing the bugs easy, once you've fixed them, consider extending the game. For example, the bomb does not obey any reasonable physics at the moment. The game would be better if it fell in a more realistic manner. There will be a separate submission area on Moodle for game extensions. You won't get any extra marks, but we are curious what you can achieve.