Design & Professional Skills
Assignment 3: Debugging the *Frogger* Game

Instructions

In the `mhandley/ENGF34-2023` github repository, in the `assignments/assignment3` directory, there are a number of python and PNG or GIF files that together comprise an implementation of the 1980s video game *Frogger*. The game is written in an object oriented style, and roughly conforms to a Model/View/Controller design pattern, as discussed in class. Run the game by typing:

```
python3 frogger.py
```

or the equivalent on your system.

The objective of the game is to cross the road, avoiding the cars, cross the river by jumping on the logs or turtles, and make it to one of the five frog homes at the top of the screen. You start out with seven lives and a certain amount of time. You must get frogs to all five homes before the time runs out, or it's Game Over. If you succeed in filling all five homes, you get to move on to the next level, and the game gets a little harder.

The game requires python 3 and a recent version of tkinter.

The problem is that the game has at least five bugs that make it unplayable.

Your Task

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.

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).

You may however want to temporarily change the code to make it easier to reproduce a bug, and then revert those changes after you've fixed the bug. A 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.

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. One will require you read the code very carefully.

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, your brief explanations of the causes of the bugs, and the python source code of your fixed version of the game.

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, in the original 1980s game, the turtles submerge every so often, and there are crocodiles and other hazards. Also the way the game gets harder with each level is somewhat better.

There will be a separate submission area on Moodle for game extensions. You won't get any extra marks, but if we like your code enough to use next year, we'll figure out some reward.