Looking back at Assignment 2

Getting tippy's apply_move right

- Our type contract said:

```
'''(GameState, Move) -> GameState

Return the new game state reached by
applying move to state self, or None if the
move is illegal.
'''
```

  This means don't change the GameState.
- To avoid inadvertently changing it, we must be careful in how we copy the old GameState.
  - We must make a copy at every level.
  - This is called making a deep copy.

One implementation of minimax

Testing minimax

- Here's what a call looks like:

```
sm = StrategyMinimax()
sm.suggest_move(SubtractSquareState(?, ?))
```

- What values should we test it on?
- [worksheet]

Increasing our confidence

- If a test case passes, does it mean minimax worked?
- If I provided a fraudulent implementation, could it pass? How could the test result tell the difference?

Our unittest cases for minimax

- For each test case, there is a probability that it could be passed by a random strategy.
- Those probabilities are:
  `0.5, 0.5, 0.5, 0.5, 0.5, and 0.2`
- But these probabilities multiply together.
- The probability of passing them all by a random strategy is
  `0.5 x 0.5 x 0.5 x 0.5 x 0.5 x 0.2`
  `= 0.00625`

About unittest

- Create a subclass of `unittest.TestCase`.
- For each test case, define a method whose name begins with `test`.
- In each test method, set up and run a test, and assert what must be true if it ran correctly.
- For any set up that all test cases need, override method `setUp` and do it there.
  - It is called before every test case.
  - The environment it creates is called a fixture.
- Similar for `tearDown` which is called *after*.