This assignment's purpose was to introduce us to value iteration. The code shown below iterates through all the states 1000 times to converge on values to assign as constants to each state.

Note: Bill and I worked on this together after class, so our code is identical. Rest assured that we all contributed equally to the assignment.

```
class ValueIteration:

     def __init__(self):

          # states
          self.states = ["s1", "s2", "s3", "s4"]
          self.actions = ["n", "p"]

          # instant reward
          self.R = {}
          self.R["s1"] = 50           # c++ h++
          self.R["s2"] = 0            # c++ h--
          self.R["s3"] = 100          # c-- h++
          self.R["s4"] = 10           # c-- h--


          # discount factor
          self.gamma = 0.01

          self.P = {}

          # current state, action => [(future state, probability), ...]
          self.P[("s1", "n")] = [("s1", 0.9), ("s2", 0.1)]
          self.P[("s1", "p")] = [("s3", 1)]

          self.P[("s2", "n")] = [("s1", 0.5), ("s2", 0.5)]
          self.P[("s2", "p")] = [("s4", 1)]

          self.P[("s3", "n")] = [("s1", 1)]
          self.P[("s3", "p")] = [("s3", 0.5), ("s4", 0.5)]

          self.P[("s4", "n")] = [("s2", 0.5), ("s3", 0.5)]
          self.P[("s4", "p")] = [("s4", 1)]

          # future reward
          self.U = {}
          self.U["s1"] = 0
```

```
            self.U["s2"] = 0
            self.U["s3"] = 0
            self.U["s4"] = 0


      def iterate(self):
            # hold the current iteration's values
            temp = {}

            for state in self.states:
                  # no chemo
                  n = self.P[(state, "n")]
                  n_val = 0
                  for future_state,future_prob in n:
                        n_val += future_prob * self.U[future_state]

                  # yes chemo
                  p = self.P[(state, "p")]
                  p_val = 0
                  for future_state,future_prob in p:
                        p_val += future_prob * self.U[future_state]

                  best = self.R[state] + self.gamma*max(n_val, p_val)
                  temp[state] = best

            self.U = temp

if __name__ == '__main__':
      v = ValueIteration()
      for i in range(0, 1000):
            v.iterate()
      print v.U
```

We found that as gamma decreases, the number of iterations it takes to converge also decreases. However, as gamma increases, the values of the constants also increase, since it is the discount factor. We also found that as transition probabilities change, the values of the constants change in or out of favor of specific states. If a certain state only has a 10% chance of being entered from any other state, then it's going to be less likely to enter, and will end up giving less benefit to the system.

Here is the first 16 iterations and the constant values stored by each state. You can see that they quickly converge, and the last 985 iterations are not really needed.

```
0 {'s3': 100.0, 's2': 0.0, 's1': 50.0, 's4': 10.0}
1 {'s3': 105.5, 's2': 2.5, 's1': 60.0, 's4': 15.0}
```

```
2 {'s3': 106.025, 's2': 3.125, 's1': 60.55, 's4': 15.4}
3 {'s3': 106.07125, 's2': 3.18375, 's1': 60.6025, 's4': 15.4575}
4 {'s3': 106.0764375, 's2': 3.1893124999999998, 's1': 60.607125, 's4': 15.46275}
5 {'s3': 106.076959375, 's2': 3.1898218750000003, 's1': 60.60764375, 's4': 15.4632875}
6 {'s3': 106.07701234375, 's2': 3.18987328125, 's1': 60.6076959375, 's4': 15.463339062500001}
7 {'s3': 106.0770175703125, 's2': 3.1898784609375, 's1': 60.607701234375, 's4': 15.46334428125}
8 {'s3': 106.07701809257813, 's2': 3.1898789847656253, 's1': 60.60770175703125, 's4': 15.463344801562501}
9 {'s3': 106.07701814470703, 's2': 3.189879037089844, 's1': 60.60770180925781, 's4': 15.463344853867188}
10 {'s3': 106.0770181499287, 's2': 3.189879042317383, 's1': 60.6077018144707, 's4': 15.463344859089844}
11 {'s3': 106.07701815045093, 's2': 3.1898790428394044, 's1': 60.60770181499287, 's4': 15.463344859612306}
12 {'s3': 106.07701815050316, 's2': 3.1898790428916137, 's1': 60.60770181504509, 's4': 15.463344859664517}
13 {'s3': 106.07701815050838, 's2': 3.1898790428968353, 's1': 60.60770181505032, 's4': 15.46334485966974}
14 {'s3': 106.0770181505089, 's2': 3.189879042897358, 's1': 60.60770181505084, 's4': 15.463344859670261}
15 {'s3': 106.07701815050896, 's2': 3.18987904289741, 's1': 60.60770181505089, 's4': 15.463344859670315}
```