```
//May 2, 2006
//shutbox_dynamic.cpp
//Implements "one die" option and prints out table of "continuing moves".

#include <iostream>
#include <iomanip.h>
#include <math.h>

#define NTILES 9
#define NBOARDS 512
#define MAXSCORE 45
#define DICEROLLS 13

int theSortedBoards[NBOARDS];
int Value[NBOARDS];
int endPosition[MAXSCORE+1];
int bestMove[NBOARDS][DICEROLLS];
double EValue[NBOARDS];

int min(int x, int y) {
    if (x <= y) return x;
    return y;
}

inline int boardValue (int theBoard) {
   int value=0;
   int temp,temp2;
   for (int i=0; i<=8; i++) {
      temp=pow(2,i);
      temp2=theBoard&temp;
      if (temp2!=0) {
         value=value+(i+1);
      }
   }
   return value;
} // end function

inline void printBoard (int theBoard) {
   int temp,temp2;
   for (int i=0; i<=8; i++) {
      temp=pow(2,i);
      temp2=theBoard&temp;
      if (temp2!=0) {
```

```
         std::cout<< "#";
      }
      else {
         std::cout << (i+1);
      }
   }
} // end function

inline void sortBoards (void) {
   int num=0; // num is the index for sorted boards
   for (int i=45; i>=0; i--) {
      for (int j=0; j<NBOARDS; j++) {
         if (boardValue(j)==i) {
            theSortedBoards[num]=j;
            Value[j]=i;
            num=num+1;
         }
      }
   endPosition[i]=num-1;
   }
   endPosition[46]=-1;
} //end function

inline void initializeBestMoves (void) {
   for (int boardNumber=0; boardNumber<NBOARDS; boardNumber++) {
      for (int diceRoll=1; diceRoll<=12; diceRoll++) {
         bestMove[boardNumber][diceRoll]=-999;
      }
   }
} // end function

inline void initializeExpectedValues (void) {
   for (int i=0; i<NBOARDS; i++) {
      EValue[i]=0;  //Initially set the probability associated with each board to 0.
  } // end for loop
  EValue[NBOARDS-1]=1; //The board 511 is 111111111. The probability of "shutting the box" from this board is 1.
} //end function

inline void calculateExpectedValue (int boardNumber) {
   int newTileSum=0;
   double bestEValue=0.0;
   double tempEValue=0.0;
   int theBestMove;
```

```
   int test1, test2;
   int minRoll=2;
   int maxRoll=12;
   if (MAXSCORE-Value[boardNumber]<=6) {  //If the sum of the unflipped tiles <=6, we roll one die
      minRoll=1;
      maxRoll=6;
   }
   for (int diceRoll=minRoll; diceRoll<=min(MAXSCORE-Value[boardNumber],maxRoll); diceRoll++) {
      newTileSum=Value[boardNumber]+diceRoll;
      theBestMove = -999;
      for (int i=endPosition[newTileSum+1]+1; i<=endPosition[newTileSum]; i++) {
      // Loop through all boards with value = newTileSum.  These boards represent all possible subsequent boards
         test1=(theSortedBoards[i]|boardNumber)==theSortedBoards[i];
         test2=(boardNumber&theSortedBoards[i])==boardNumber;
         if (test1*test2==1) {
            if (minRoll==2) {
               tempEValue=EValue[theSortedBoards[i]]*(6-abs(diceRoll-7))/36.0;
            }
            else {
               tempEValue=EValue[theSortedBoards[i]]*(1/6.0);
            }
            if (tempEValue>bestEValue) {
                  bestEValue=tempEValue;
                  theBestMove=theSortedBoards[i];
            } //end if
          } //end if
       } //end for
       if (bestEValue!=0) {
          EValue[boardNumber]=EValue[boardNumber]+bestEValue;
       }
       bestMove[boardNumber][diceRoll]=theBestMove;
       bestEValue=0;
       tempEValue=0;
   } //end for loop
} //end function

inline void processBoards (int maxBoard) {
//We process the boards starting with those with lowest tile sums, working our way "up".  We rely
//on previous calculations to determine expected values for various boards.
   for (int i=0; i<maxBoard; i++) {
      calculateExpectedValue(theSortedBoards[i]);
    }
}
```

```
inline void outputProbabilities (int maxBoard) {
   std::cout.setf(ios::showpoint);
   std::cout.precision(6);
   std::cout.setf(ios::left);
   std::cout << "\t\t\t\t\t\t\t\t\t\t\t\t\t" << "BEST MOVE TO CONTINUE GAME WHEN n IS ROLLED\n";
   std::cout << setw(10) << "Config" << setw(10) << "P(Shut)" << setw(10) << "\tn=1" << setw(10) << "\tn=2";
   std::cout << setw(10) << "\tn=3" << setw(10) << "\tn=4" << setw(10) << "\tn=5"<< setw(10) << "\tn=6" << setw(10);
   std::cout << "\tn=7" << setw(10) << "\tn=8" << setw(10) << "\tn=9" << setw(10) << "\tn=10" << setw(10) << "\tn=11";
   std::cout << setw(10) << "\tn=12" << "\n";
   std::cout.setf(ios::fixed);
   for (int i=0; i<maxBoard; i++) {
      printBoard(theSortedBoards[i]);
      std::cout << "\t";
      std::cout << setw(10) << EValue[theSortedBoards[i]] << "\t";
      for (int j=1; j<=12; j++) {
         if (bestMove[theSortedBoards[i]][j]==-999) {
            std::cout << "---------";
         }
         else {
            printBoard(bestMove[theSortedBoards[i]][j]);
         }
         std::cout << "\t";
      }
      std::cout << "\n";
  } // end for loop
} // end function

int main (int argc, char * const argv[]) {
    initializeExpectedValues();
    initializeBestMoves();
    sortBoards();
    processBoards(NBOARDS);
    outputProbabilities(NBOARDS);
    return 0;
}
```