```
allRolls :: DiceChoice
         -> (DiceVals, Integer)

         -> [ (DiceVals, Integer) ]
allRolls choices (vs, n) = case pop choices vs of
  Nothing -> [ ([], n-1) ]
  Just ((chosen, v), (choices, vs)) ->
    allRolls choices (vs, error "Didn't expect to use"
      >>= \(roll, _) -> [ (d:roll,  n-1)
        | d <- rollList ]
        where
          rollList = if chosen then [v] else [ 1..6 ]

example =
  let diceChoices = [ False, True, True, False, False
      diceVals = [ 6, 4, 4, 3, 1 ]
  in mapM_ print $ allRolls diceChoices (diceVals, 2)
```

```
allRolls :: DiceChoice
         -> DiceVals
         -> Integer
         -> [ (DiceVals, Integer) ]
allRolls choices vs n = case pop choices vs of
   Nothing -> [ ([], n-1) ]
   Just ((chosen, v), (choices, vs)) ->
    allRolls choices vs (error "Didn't expect to use")
      >>= \(roll, _) -> [ (d:roll,  n-1)
        | d <- rollList ]
        where
          rollList = if chosen then [v] else [ 1..6 ]

example =
  let diceChoices = [ False, True, True, False, False
      diceVals = [ 6, 4, 4, 3, 1 ]
  in mapM_ print $ allRolls diceChoices diceVals 2
```