Following is the program running, note that the board only starts 1 move from the solution. I play with it a bit then solve.

```
Current position:
1 2
345
678
Your move: 3
Current position:
12
345
678
Your move: 6
Current position:
125
34
678
Your move: 3
Current position:
12
345
678
Your move: 2
Current position:
1 2
345
678
Your move: 1
 12
345
678
Solved!
```

Now the scala code: Game.scala:

```
import puzzel.Puzzel
import scala.io.StdIn.{readLine,readInt}


object Game {
    def main(args: Array[String]): Unit = {
        while(!Puzzel.isSolved()) {
                println("Current position:")
                Puzzel.PrintBoard()
                print("Your move: ")
            var input = readInt()
            Puzzel.MoveTile(input)
```

```
        }
        Puzzel.PrintBoard()
        println("Solved!")
    }
}
```

Puzzel.scala

```
package puzzel

import Math.{sqrt, pow}

object Puzzel {
    var Solution = Array.ofDim[Int](3,3)
    var TileBoard = Array.ofDim[Int](3,3)
    var indexer = 0

    for (
        i <- 0 until 3;
        j <- 0 until 3
    ) {
        Solution(i)(j) = indexer
        TileBoard(i)(j) = indexer
        indexer += 1

    }

    RandomizeTable()

    /* RandomizeTable() Doesn't actuall randomize table, because
       it's possible
     * to put together a table that isn't solvable. This just
     * moves the tiles around a bunch.
     */
    private def RandomizeTable() = {
        TileBoard(0)(0) = 1
        TileBoard(0)(1) = 0
    }

    /* SanityCheck() just makes sure the user made a valid choice
       */
    private def SanityCheck(tileChoice: Int): Double = {
        var xVal: Int = 0
        var yVal: Int = 0

        var dxVal: Int = 0
```

```
var dyVal: Int = 0

var zeroTuple = searchArrays(0) // Get the zero coord.
xVal = zeroTuple(0)._1
yVal = zeroTuple(0)._2

tileChoice match {
    case 1 => {
        dxVal = 0
        dyVal = 0
    }
    case 2 => {
        dxVal = 0
        dyVal = 1
    }
    case 3 => {
        dxVal = 0
        dyVal = 2
    }
    case 4 => {
        dxVal = 1
        dyVal = 0
    }
    case 5 => {
        dxVal = 1
        dyVal = 1
    }
    case 6 => {
        dxVal = 1
        dyVal = 2
    }
    case 7 => {
        dxVal = 2
        dyVal = 0
    }
    case 8 => {
        dxVal = 2
        dyVal = 1
    }
    case 9 => {
        dxVal = 2
        dyVal = 2
    }
}
sqrt(pow((dxVal - xVal),2) + pow((dyVal - yVal),2)) //
```

```
        Calculate distance
}

/* SearchArray(INT) will find the position of a given
 * tile in the board to get the coordinates,
 * returning the 2 tuple of the coords.
 */
private def searchArrays(lookup: Int ) =
    for {
        i <- 0 until TileBoard.size
        j <- 0 until TileBoard(i).size
        if TileBoard(i)(j) == lookup
  } yield (i, j)

  /* MoveTile(INT) takes in a tile to
   * move and checks the move is valid
   * and updates the board with the move.
   */
def MoveTile (tile: Int): Unit = {

    if (SanityCheck(tile) > 1) { // Check for valid tile move
          println("ERROR invalid move!")
          return
        }
    val tmpCord = searchArrays(0) // Look for zero tile, our
       empty space

    tile match {
        case 1 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
           TileBoard(0)(0)
                    TileBoard(0)(0) = 0 }
        case 2 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
           TileBoard(0)(1)
                    TileBoard(0)(1) = 0 }
        case 3 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
           TileBoard(0)(2)
                    TileBoard(0)(2) = 0 }

        case 4 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
           TileBoard(1)(0)
                    TileBoard(1)(0) = 0 }
        case 5 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
           TileBoard(1)(1)
                    TileBoard(1)(1) = 0 }
        case 6 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
```

```
                TileBoard(1)(2)
                        TileBoard(1)(2) = 0 }

            case 7 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
                TileBoard(1)(0)
                        TileBoard(1)(0) = 0 }
            case 8 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
                TileBoard(1)(1)
                        TileBoard(1)(1) = 0 }
            case 9 => { TileBoard(tmpCord(0)._1)(tmpCord(0)._2) =
                TileBoard(1)(2)
                        TileBoard(1)(2) = 0 }
        }
    }

    /* isSolved() will return true if the
     * puzzel is solved
     */
    def isSolved () : Boolean = {
        if (Solution.deep == TileBoard.deep) return true
        else return false
    }

    /* Prints the board! */
    def PrintBoard () = {
        for (
            i <- 0 until 3;
            j <- 0 until 3
        ) yield {
            if (TileBoard(i)(j) == 0 ) print("_")
            else print(TileBoard(i)(j))
            if(j == 2) println()
        }
    }

}
```