Ficha 6

Exercicio 6.1

a)

Dentro do def mainLoop(...) no userInput match {} Introduzir o seguinte case:

```
case "N" | "n" => {
    mainLoop(GameState(0, 0), random)
}
```

b)

In CoinFlip object

Create:

```
val history = List():List[GameState]
```

Change mainLoop:

```
mainLoop(s, r, history)
```

```
def mainLoop(gameState: GameState, random: Random, history: List[GameState])
{...}
```

Change the match case:

```
case "N" => {
    val newHistory = history ++ List(gameState)
    mainLoop(GameState(0, 0), random, newHistory)
}
```

And:

```
case _    => {
    printGameOver()
    printGameState(gameState)
    printGameStateHistory(history)
    // return out of the recursion here
}
```

In CoinFlipUtils object

```
def printGameStateHistory(lst: List[GameState]): Unit = {
    println("History: ")
    lst map (x => printGameState(x) + "\n")
}
```

c)

In CoinFlip object

```
val r = MyRandom(11)
```

Change mainLoop:

```
def mainLoop(gameState: GameState, random: RandomWithState, history:
List[GameState]) {...}
```

Inside the match case:

```
userInput match {
  case "H" | "T" => {
    val coinTossResult = tossCoin(random)
    val newNumFlips = gameState.numFlips + 1
  if (userInput == coinTossResult._1) {
      val newNumCorrect = gameState.numCorrect + 1
   val newGameState = gameState.copy(numFlips = newNumFlips, numCorrect =
newNumCorrect)
      printGameState(printableFlipResult(coinTossResult._1), newGameState)
      mainLoop(newGameState, coinTossResult._2, history)
    } else {
      val newGameState = gameState.copy(numFlips = newNumFlips)
      printGameState(printableFlipResult(coinTossResult._1), newGameState)
      mainLoop(newGameState, coinTossResult._2, history)
```

In CoinFlipUtils object

```
def tossCoin(r: RandomWithState): (String, RandomWithState) = {
    val (i, state) = r.nextInt(2)
  i match {
        case 0 => ("H", state)
        case 1 => ("T", state)
    }
}
```

6.2

In the Main.scala file inside the match case:

```
case "C" => {
  IO_Utils.showOptionsNum()
  val userInputNum = IO_Utils.getUserInput()

  IO_Utils.showOptionsNP()
  val userInputNP = IO_Utils.getUserInput()

  if (IO_Utils.myToInt(userInputNum) != None) {
    val newState = state.changeNP(IO_Utils.myToInt(userInputNum).get,
IO_Utils.toNT(userInputNP))
    IO_Utils.printTurmaState(newState)
    mainLoop(newState)
  }
  else {
    IO_Utils.showPrompt("Wrong number!")
    mainLoop(state)
  }
}
```

6.3

in the main object:

```
val options = SortedMap[Int, CommandLineOption](
  1 -> new CommandLineOption("Add", Container.addEntry(IO_Utils.prompt("Key"),
IO_Utils.prompt("Value"))),
  2 -> CommandLineOption("Remove", Container.removeEntry(IO_Utils.prompt("Key"))),
  3 -> CommandLineOption("Update", Container.updateEntry(IO_Utils.prompt("Key"),
IO_Utils.prompt("Value"))),
  4 -> CommandLineOption("Show map content", IO_Utils.printContainer),
  0-> new CommandLineOption("Exit", _ => sys.exit)
)
```

remove an Entry:

```
def removeEntry(key: => String)(container: Container): Container =
{
  new Container(container.name, container.data - key)
}
```

update an Entry:

```
def updateEntry(key: => String, value: =>String)(container: Container): Container =
{
  new Container(container.name, container.data + (key -> value))
}
```

show the content of the map:

update printContainer method

```
def printContainer(container: Container) = {
  println("Name:"+container.name)
  container.data.toList map (x => println("Key:"+x._1) + " " + println("Value:"+x._2))
  container
}
```