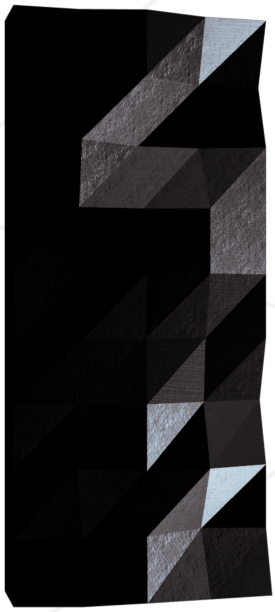

Go Piscine

Rush 00

Summary: This document is the subject for Rush00 of the Go Piscine @ 42Tokyo.

Contents

I	**Instructions**	**2**
II	**Rush00**	**4**
III	**Bonus**	**7**

Chapter I

Instructions

- Each member of the group can register the whole group to defense.
- The group MUST be registered to defense.
- Any question concerning the subject would complicate the subject.
- You have to follow the submission procedures for all your exercises.
- This subject could change up to an hour before submission.
- You will have to handle errors coherently. Feel free to either print an error message, or simply return control to the user.
- Rushes exercises have to be carried out by group of 2, 3 or 4.
- You must therefore do the project with the imposed team and show up at Your defense slot, with all of your teammates.
- You project must be done by the time you get to defense. The purpose of defense is for you to present and explain your work.
- Each member of your group must be fully aware of the works of the project. Should you choose to split the workload, make sure you all understand what everybody's done. During the defense, you'll be asked questions and the final grade will be based on the worst explanations.
- It goes without saying, but gathering the group is your responsibility. You've got all the means to get in contact with your teammates: phone, email, carrier pigeon, spiritism, etc. So don't bother blurping up excuses. Life isn't fair, that's just the way it is.
- However, if you've really tried everything one of your teammates remains unreachable: do the project anyway, and we'll try and see what we can do about it during the defense. Even if the group leader is missing, you still have access to the submission directory.
- You must use the latest version of Go.
- Your turn-in directory for each exercise should look something like this:

```
ex[XX]
|-- main.go
|-- vendor
   |-- ft
      |-- printrune.go
   |-- piscine
      |-- *.go
```

Make sure the subject that was originally assigned to your group works perfectly before considering bonuses: If a bonus subject works, but the original one fails the tests, you'll get 0.

Chapter II

Rush00

	Exercise 00
Rush00	
Turn-in directory : $ex00/$	
Files to turn in : `*`	
Allowed packages : `None`	
Allowed builtin functions : `None`	

Write a function that takes rows of a chessboard as arguments and checks if your King is "in check".

- Quick reminder, chess is a game played on a chessboard, an 8 smaller-square long square board with specific pieces: King, Queen, Bishop, Knight, Rook and Pawns. For this exercice, you will only play with Pawns, Bishops, Rooks, Queens... and a King.
- A piece can only capture the first possible piece that stands on its path.
- The board can be of different sizes but will remain a square. There's only one King and all other pieces are against it. All the characters that are not used to refer to pieces are considered as empty squares.
- The King is considered to be "in check" when an other enemy piece can capture it. When it's the case, you will print "Success" on the standard output followed by a newline, otherwise you will print "Fail" followed by a newline.
- Your function should never crash or loop indefinitely.
- Whenever an undefiend behavior occurs you should return an error message or prints nothing and gives back control.
- Your `main` will be modified during defense, to check if you've handled everything you're supposed to. Here's an example of test we'll perform:

Example1)

```
package main

func main() {
        board := []string{
                "R...",
                ".K..",
                "..P.",
                "....",
        }

        checkmate(board)
}
```

Example2)

```
package main

func main() {
        board := []string{
                "..",
                ".K",
        }

        checkmate(board)
}
```

- Here's an example of how we run the test.

```
$> go mod init ex00
$> go run . | cat -e
Success$
```

- Each piece has specific moves available, and all patterns to capture enemy pieces are detailed below.

```
Pawn (P):

. . . . . . .
. . . . . . .
. . X . X . .
. . . P . . .
. . . . . . .
. . . . . . .
. . . . . . .

Bishop (B):

X . . . . . X
. X . . . X . .
. . X . X . . .
. . . B . . . .
. . X . X . . .
. X . . . X . .
X . . . . . X

Rook (R):

. . . X . . .
. . . X . . .
. . . X . . .
X X X R X X X
. . . X . . .
. . . X . . .
. . . X . . .

Queen (Q)

X . . X . . X
. X . X . X . .
. . X X X . . .
X X X Q X X X
. . X X X . . .
. X . X . X . .
X . . X . . X
```

Chapter III

Bonus

	Exercise 01
Bonus	
Turn-in directory : *ex*01/	
Files to turn in : *	
Allowed packages : Everything except external packages	
Allowed builtin functions : *	

Add more functionality to ex00.
For example:

- Create a program that accept takes rows of a chessboard as arguments and checks if your King is "in check".

```
$> go mod init ex01
$> go run . 'R...' '.K..' '..P.' '....' | cat -e
Success$
$> go run . '..' '.K' | cat -e
Fail$
```

- Create a program that takes the best move for a checkmate.
- Create a chessgame.

Other (creative) functionality will be graded too.
For each functionality, 5 points are given. (Max 25points)