

1 Project Description

In this assignment there are 4 parts. For each part you should:

- Write the appropriate code.
- Include comments within the code to explain the algorithm.
- Test the code to ensure its correctness.
- Format and structure the code to maximise its readability.

A report must be submitted containing a cover page, the solutions to each of the four parts, and your code, as a PDF, to the vUWS submission site. The cover page must contain your name, student number, unit number and name, and the declaration below.

Submission is due by Friday of week 13. Late submissions will receive a 10% reduction in marks for each day late.

2 Marking Criteria

This assignment is worth 40% of the unit assessment tasks. There four problems to investigate and 10 marks available for each of the four problems. The marking criteria for each question is given in Table 1.

Criteria	Q1	Q2	Q3	Q4
Code Correctness (5 marks)				
Comments explaining code (2 marks)				
Code Testing (1 mark)				
Code Style and Readability (2 marks)				
Total (10 marks)				

Table 1: Marking criteria for each part of this project.

When writing the solutions to each of the four parts, make sure to consult the marking criteria and check that you have covered them. The project will be marked using this criteria.

3 Declaration

Before submitting the assignment, include the following declaration in a clearly visible and readable place on the cover page of your project report.

By including this statement, we the authors of this work, verify that:

- We hold a copy of this assignment that we can produce if the original is lost or damaged.
- We hereby certify that no part of this assignment/product has been copied from any other student's work or from any other source except where due acknowledgement is made in the assignment.
- We are aware that this work may be reproduced and submitted to plagiarism detection software programs for the purpose of detecting possible plagiarism (**which may retain a copy on its database for future plagiarism checking**).
- We hereby certify that we have read and understand what the School of Computing, Engineering and Mathematics defines as minor and substantial breaches of misconduct as outlined in the learning guide for this unit.

Note: An examiner or lecturer/tutor has the right not to mark this project report if the above declaration has not been added to the cover of the report.

4 Project Tasks

Ultra Indoor Cricket is a new form of cricket that has grown from a form of backyard cricket, where each team consists of one player, but many teams can play at once (at least two teams are needed).

- At any time, there is one bowler and one batsman and any other players are fielders.
- The batsman bats for a set number of overs. where each over consist of six balls and each of the overs to be bowled are evenly divided between the remaining players (e.g. 6 overs for one player, 3 for two, 2 for three).
- Due to the game being indoor, no more that one run can be run, but 2, 3, and 5 runs can be awarded for hitting certain targets.
- If a batsman is given out, they lose five runs, and the batsman continues to bat (so it is possible to finish with a negative number of runs).
- Finally, any illegal deliveries bowled subtract one run from bowler's batting score and add three to the batsman's score; a batsman cannot not score runs from an illegal delivery and unlike other forms of cricket an extra ball is not bowled.

The winner is the player with the most runs after all players have batted.

The Ultimate Indoor Cricket League want you to write a program to help keep track of games. They have asked you to complete the following tasks.

4.1 Initialising Player State

For each player, we must record the player's name, the number of times they scored 0, 1, 2, 3 and 5 runs, the number of illegal balls bowled, and the number of times they were given out.

Write a function `initialisePlayer` that takes a player name, and creates and returns a data structure (such as a list) that provides a player's state at the beginning of the game (with all counts set to zero).

Write another function `playerScoreBoard` that takes a player's data structure and prints out the contents of the list and the players total number of runs (calculated from the list contents) to provide a scoreboard for the player.

The code below shows the how the functions should be called and the desired output.

```
playerA <- initialisePlayer("T Smith")
playerB <- initialisePlayer("N Starc")
playerScoreBoard(playerA, header = TRUE)
```

Name	0s	1s	2s	3s	5s	X	Out	Tot
T Smith	0	0	0	0	0	0	0	0

```
playerScoreBoard(playerB)
```

N Starc	0	0	0	0	0	0	0	0
---------	---	---	---	---	---	---	---	---

Show that your code produces the same results.

4.2 Updating the Player State

An over consists of six balls bowled by one bowler to one batsman and the outcome of each ball can be one of 0 runs, 1 run, 2 runs, 3 runs, 5 runs, illegal, or out. The batsman's state must be updated after each of the balls, but the bowlers state is only adjusted after an illegal ball.

Write a function `updateBatsman` that takes the outcome of a ball and the batsman's state, and returns the batsman's updated state.

Write a function `updateBowler` that takes the outcome of a ball and the bowler's state, and returns the bowler's updated state. Remember that the bowler's state is only effected by illegal balls.

The code below shows the how the functions should be called and the desired output.

```
ballOutcomes <- c("run0", "run1", "run2", "run3", "run5", "illegal", "out")
```

```
# test batsman
```

```
playerScoreBoard(updateBatsman("run0", playerA), header = TRUE)
```

Name	0s	1s	2s	3s	5s	X	Out	Tot
T Smith	1	0	0	0	0	0	0	0

```
playerScoreBoard(updateBatsman("run1", playerA))
```

T Smith	0	1	0	0	0	0	0	1
---------	---	---	---	---	---	---	---	---

```
playerScoreBoard(updateBatsman("out", playerA))
```

```

T Smith  0  0  0  0  0  0  1 -5
playerScoreBoard(updateBatsman("illegal", playerA))

```

```

T Smith  0  0  0  1  0  0  0  3
# test bowler
playerScoreBoard(updateBowler("run0", playerA))

```

```

T Smith  0  0  0  0  0  0  0  0
playerScoreBoard(updateBowler("run5", playerA))

```

```

T Smith  0  0  0  0  0  0  0  0
playerScoreBoard(updateBowler("illegal", playerA))

```

```

T Smith  0  0  0  0  0  1  0 -1

```

Show that your code produces the same results.

4.3 One Over

Write the function `generateOver` to randomly sample from the above ball outcomes to provide the outcome of six balls. Run the function and show the result of the over.

Write the functions `updateBatsmanOver` and `updateBowlerOver` that takes the outcome of six balls, the state of the batsman and the state of the bowler before the over begins, and returns the state of the batsman and bowler after the over ends, respectively.

Use the functions to generate an over, update the batsman and bowler states based on the over and print out the final states using the `playerScoreBoard` function.

The code below shows the how the functions should be called and the desired output.

```

over <- generateOver(ballOutcomes)
print(over)

[1] "run1"    "out"     "illegal" "run2"    "run0"    "run1"

playerScoreBoard(updateBatsmanOver(over, playerA), header = TRUE)

  Name 0s 1s 2s 3s 5s  X Out Tot
T Smith  1  2  1  1  0  0  1  2

playerScoreBoard(updateBowlerOver(over, playerB))

```

```

N Starc  0  0  0  0  0  1  0 -1

```

Show that your code produces the equivalent results.

4.4 Match Result

The official Ultimate Indoor Cricket rules provide a sequencing for the bowler and batter for each over. Below we show the sequence for a two and three player game containing six overs. The variable `over` provides the over number, the variables `bowler` and `batter` provide the id of the player.

```
twoPlayer <- list(  
  ## batter and bowler for each over  
  list(over = 1, bowler = 1, batter = 2),  
  list(over = 2, bowler = 2, batter = 1),  
  list(over = 3, bowler = 1, batter = 2),  
  list(over = 4, bowler = 2, batter = 1),  
  list(over = 5, bowler = 1, batter = 2),  
  list(over = 6, bowler = 2, batter = 1)  
)
```

```
threePlayer <- list(  
  ## batter and bowler for each over  
  list(over = 1, bowler = 1, batter = 2),  
  list(over = 2, bowler = 2, batter = 3),  
  list(over = 3, bowler = 3, batter = 2),  
  list(over = 4, bowler = 2, batter = 1),  
  list(over = 5, bowler = 1, batter = 3),  
  list(over = 6, bowler = 3, batter = 1)  
)
```

```
## choose two or three player  
#overSequence <- twoPlayer  
overSequence <- threePlayer
```

The following list shows “T Smith” with id 1, “B Border” with id 2, and “S Ponting” with id 3.

```
players <- list(  
  initialisePlayer("T Smith"),  
  initialisePlayer("B Border"),  
  initialisePlayer("S Ponting")  
)
```

Source the file `uicMatchOversOutcome.R` to obtain a list called `gameOvers` with the outcome of six overs of an official UIC match (each list element is one over). Then write the code to compute the player states after playing the game using the sequence given by the chosen over sequence. The code should provide results using either the two player or three player sequence (the code should work using either). Show the game results using the `playerScoreBoard` function.

The code below shows the player scores after a game.

```
playerScoreBoard(players[[1]], header = TRUE)
```

Name	0s	1s	2s	3s	5s	X	Out	Tot
T Smith	1	3	2	4	2	4	0	25

```
playerScoreBoard(players[[2]])
```

B Border	0	1	3	6	1	1	1	24
----------	---	---	---	---	---	---	---	----

```
playerScoreBoard(players[[3]])
```

S Ponting 1 1 1 3 1 1 5 -9

Show your set of player results from the game.

When writing up your report, make sure to structure and comment on your code to make it readable. Make sure to follow the marking criteria to ensure that the maximum marks are obtained.