# Assignment 3 (Loops and Lists) for COMSPCI 101: Mū Tōrere

**Due:** 4:30pm, 9th October 2020.

**Worth:** This assignment is marked out of 30 and is worth 3% of your final mark.

**Topics covered:**

Loops and lists

*The work done on this assignment must be your own work. Think carefully about any problems you come across and try to solve them yourself before you ask anyone else for help. Under no circumstances should you use code written by another person in your assignment solution or give your code to another student.*

**Very Important:**

This assignment has two sections.

**Section A** is marked using Coderunner (22 marks). For this section you need to define 11 functions. The 11 functions, when inserted into the Section B skeleton program, create a single player version of the game of Mū Tōrere (see the game description below). Each function is described in Section A of this document and there are two Python skeleton programs which you MUST use to develop the 11 functions.

**Section B** (8 marks).

Insert the 11 functions from Section A into the Assignment 3 program. The program should execute without error allowing the user to play the Mū Tōrere game.

**Submission**

Section A - Functions 1 - 11 are submitted using CodeRunner3.

Section B - Submit your completed Assignment 3 Python program (just one Python file named "YourUsernameA3.py", e.g., dwil237A3.py) using the Assignment Dropbox:

https://adb.auckland.ac.nz/Home/

**Notes:**

• This assignment is marked out of 30 and is worth 3% of your final mark. Five marks out of 30 are assigned for the style of your program (program docstring, meaningful variable names, etc.). Three marks out of 30 are assigned if your final Mū Tōrere program executes without error.

• You MUST only use the features taught in CompSci 101.

• An example output using the completed program is available at the end of this document.

## Assignment 3: The game of Mū Tōrere

In this assignment you will be completing Python functions to implement a single player version of the game *Mū Tōrere*, which means "Fast Move". This Māori game is also an opportunity to introduce you to some te reo Māori (Māori language) terms.

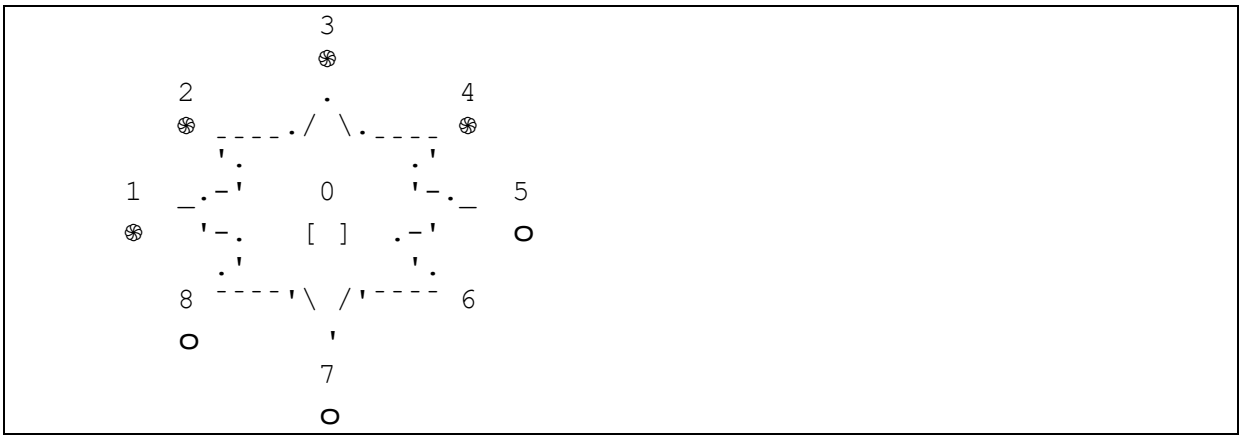
- The pronunciation of *Mū Tōrere* can be heard here: https://maoridictionary.co.nz/search?keywords=mu+torere.
- See the "Further information about Mū Tōrere" section for a deeper account of the game.

### Mū Tōrere Game Instructions

Aim: The aim of the game is to move your pieces into a configuration that blocks your opponent from moving.

Starting positions: The papa tākaro (game board) is in the shape of an eight-sided star with eight playing positions at each of the eight tips and one in the centre. There are nine positions on the board. The pūtahi (position 0) is in the centre of the board. This is surrounded by eight kēwai positions (1-8).

Each player has four perepere (play pieces). To set up the game, each player's perepere are positioned on one-half of the kēwai (external points) on the board. The pūtahi is always empty in the initial configuration. An example of the initial game positions (labelled 0 to 8) are as follows:




Player 1 (the computer) is ֍ and Player 2 (you) is ○. An empty position is represented by `[ ]`.

Rules of movement:

- You may move to an empty connected position. You cannot jump over another perepere.

- If you are in a kēwai position (1-8), you can move to an adjacent position in that range if it is unoccupied. (Note that 1 and 8 are adjacent.)

- If you are in a kēwai position (1-8) and the flanking kēwai DO NOT BOTH contain perepere from your team, you can move to the pūtahi (position 0) if it is unoccupied.

- If you are in the pūtahi position (0), you can move to a kēwai position (1-8) if it is empty.

You can see examples of Mū Tōrere being played in the following videos:

(1) https://www.youtube.com/watch?v=Y2L66uNTAFw and

(2) https://www.youtube.com/watch?v=zeh3NQbGjUg.

## *Mū Tōrere* Glossary of te reo Māori terms:

*Papa tākaro* (game board)

*Kēwai* (points): the eight positions on the outside of the game board. Sometimes called *kāwai* (tentacles). (These are referred to as positions 1 to 8 in our assignment.)

*Pūtahi* (centre position): the position in the centre of the eight outer positions on the board. (This is position 0 in our assignment.)

*Perepere* (stone playing pieces): each player has four *perepere*. There are eight *perepere* in total.

*Mā te wā.* (Until next time.)

*Ka rawe!* (Excellent!)

*Kia tūpato!* (Be careful!)

*Kia kaha!* (Be strong, keep going!)

## Assignment 3 Section A (22 marks)

For this section you need to develop 11 functions described in Section A of this document. Each of the functions are to be done in IDLE and then submitted using CodeRunner3. Download the two Python skeleton programs from the CompSci 101 assignments website:

https://www.cs.auckland.ac.nz/courses/compsci101s2c/assignments/

The following two programs must be used to develop the 11 functions used in the Mū Tōrere game.

- Use the A3Functions1To6.py program to complete and test functions 1, 2, 3, 4, 5 and 6.
- Use the A3Functions7To11.py program to complete and test functions 7, 8, 9, 10 and 11.

Once you are happy that a function executes correctly, submit it to CodeRunner:

https://coderunner3.auckland.ac.nz/moodle/

### *Section A*

The skeleton code for each of the Section A programs is provided. Each program requires you to complete the functions. Develop and test your functions in the testing programs in A3Functions1To6.py and A3Functions7To11.py. When you are happy with your code, you can further test that the function is correct by pasting the whole function—including the header—into CodeRunner3 and pressing the CHECK button.

After the description of each of the 11 functions there is a textbox with some example code and below it a textbox containing the expected output for that code.

### Q1) initialise_list. (*Initialising a list*)

Requirement: A function is required to reset the list that is used to store game moves for the purposes of replay if a new game is begun.

Assignment Task: Complete the function **initialise_list()** that returns an empty list.

Example code:

```
current_game_move = initialise_list()
print(current_game_move)
```

Output:

```
[]
```

<-----*----->

### Q2) concatenate_move. (*Concatenating lists.*)

Requirement: A function is required to concatenate the latest move to a running list of game moves (that can later be used for the purposes of replaying the game).

Assignment Task: Complete the function **concatenate_move(existing_moves, new_move)** that takes a list argument, **existing_moves**, and an integer argument, **new_move**, and returns a new list that **concatenates** two lists: **existing_moves** and **new_move** (converted to a list).

Note: You must use list concatenation to answer this question.

Example code:

```
move_list = [1,8,0,1]
move_list = concatenate_move(move_list, 8)
print(move_list)
move_list = concatenate_move(move_list, 7)
print(move_list)
```

Output:

```
[1, 8, 0, 1, 8]
[1, 8, 0, 1, 8, 7]
```

<-----*----->

### Q3) move_piece. (Update elements in a list in place)

Requirement: A function is required to update the positions of perepere on the board when a player makes a move.

Assignment Task: Complete the function **move_piece(pos_list, from_position, to_position)** that updates a list of integer values of 0, 1 and 2. The function takes **pos_list**, a list of integer elements that represent which player's piece is in each position in the board. The index value of each element in **pos_list** is the position on the board (0 to 8). The possible values of the elements in **pos_list** are

the integers 0 (empty space), 1 (player 1's perepere), and 2 (player 2's perepere). The integer **from_position** is the index value of the position in **pos_list** that a player piece is moved from. The integer **to_position** is the index value of the position in **pos_list** that a player piece is moved to.

Update **pos_list** in place by copying the element in **pos_list** at index **from_position** to the index **to_position**. Set the **pos_list** element at index **from_position** to the integer 0.

Example code:

```
position_list = [0,1,1,1,1,2,2,2,2]
move_piece(position_list, 1, 0)
print(position_list)
move_piece(position_list, 8, 1)
print(position_list)
```

Output:

```
[1, 0, 1, 1, 1, 2, 2, 2, 2]
[1, 2, 1, 1, 1, 2, 2, 2, 0]
```

<-----*----->

### Q4) find_empty_position. (Locate an element in a list.)

Requirement: A function is required to identify the empty board position.

Assignment task: Complete the function **find_empty_position(pos_list)** that returns an integer that is the index of the element in **pos_list** that has a value of 0 (indicating an empty position on the board).

Example code:

```
position_list = [0,1,1,1,1,2,2,2,2]
empty_point = find_empty_position(position_list)
print(empty_point)
position_list = [1,1,0,1,1,2,2,2,2]
empty_point = find_empty_position(position_list)
print(empty_point)
```

Output:

```
0
2
```

<-----*----->

### Q5) get_next_move. (While loop. Use in operator.)

Requirement: A function is required for the user to input the position of the piece that they wish to move (or to exit the game).

Assignment Task: Complete the function **get_next_move()** that prompts the user to enter a valid response and **returns** a string variable with the user input in uppercase when a valid response is entered by the user. Valid responses are specified in the **valid_response** list (`["0","1","2","3","4","5","6","7","8","Q","QUIT","EXIT"]`).

The user input is converted to upper case and is checked to see whether it occurs in the **valid_response** list. If the string is in the list, then the function returns a string variable with the valid response. If the string is not in the list then the function prompts the user for a valid response again until a valid response is entered. The prompt should look like the examples below.

(Note: Do not use a for…in loop in this function.)

Example code:

```
next_move = get_next_move()
print(next_move)
next_move = get_next_move()
print(next_move)
```

Output (with user input in magenta):

```
Which piece do you want to move (or Q to quit)? y
Which piece do you want to move (or Q to quit)? 9
Which piece do you want to move (or Q to quit)? 3
3
Which piece do you want to move (or Q to quit)? q
Q
```

<-----*----->

### Q6) suggest_valid_move. (Access an item from a list and assign it to a variable.)

Requirement: From a list of available moves, this function must randomly select one of those moves. This function is used for the computer player to select a move.

Assignment Task: Complete the function **suggest_valid_move(pos_list, player)** that **returns a string** that is a randomly selected element from a list.

Detailed instructions: The function **suggest_valid_move()** calls another function, **list_valid_moves(pos_list, player)**, passing it the same arguments that are passed to the **suggest_valid_move()** function, namely, **pos_list** and **player**. The **list_valid_moves()** function will return a list of string elements. (You do not need to code the function **list_valid_moves()** to answer this question. See the note below.) From the list of strings that are returned by **list_valid_moves()**, the **suggest_valid_move()** function must randomly select one string from the list and return this value. If the list returned from calling the **list_valid_moves()** function is empty then **suggest_valid_move()** will return the string "00" (i.e., two zeros).

(Note: for development purposes, the skeleton program for this question uses a program stub for the function **list_valid_moves(pos_list, player).** This means that the version of **list_valid_moves(pos_list, player)** that is listed in this skeleton program is sufficient for our testing purposes but is not to be submitted in Section B of this assignment.)

Example code:

```
get_move = suggest_valid_move([1,1,0,1,2,2,1,2,2], 1)
print(get_move)
get_move = suggest_valid_move([1,1,0,1,2,2,1,2,2], 2)
print(get_move)
```

Output (note that this is an example only. Actual results may differ because of random selection.):

```
02
00
```

<-----*----->

## Q7) new_game_positions. (Creating a new list. For…in range() loop. Returning a list with specified ordering. Appending list items.)

Requirement: Players will not want to start in the same positions each time. Create a function that generates a different configuration of starting positions that conform to the rules of the game.

Assignment Task: Create a function **new_game_positions(start_index)** that takes an integer value specifying the position of the first Player 1 piece in the starting sequence of four Player 1 pieces. **new_game_positions() returns** a list of integers that represent what kind of piece is in each position. Remember that positions 1 to 8 are in a circle on the game board so, for example, calling this function with a **start_index** of 6 will return the list [0,1,2,2,2,2,1,1,1]. The list object that is returned by the function must satisfy the following criteria:

- The list contains nine elements: one element for each position in the board. The index number of each element is the corresponding position on the board. (For example, the list element with the index of 0 will contain information regarding the centre position on the game board.)
- In each new game, the central position is empty. To represent this, the very first element of the generated list must always contain the integer zero (0), which represents an empty space in the centre of the board.
- The next eight list elements will either be an integer value of 1 or of 2. (These integer values represent whether Player 1 or Player 2 have their playing piece in that position on the board, respectively.)
- The list will always contain one 0, four 1s and four 2s as integer elements in the list (though the exact order will be specified by the value of the **start_index** argument).
- The list itself must contain four consecutive elements with an integer value of 1 and/or four consecutive elements with an integer value of 2.

(Note: you must use a **for…in range()** loop in this function. When adding the elements to the list that represent the starting positions for Player 1 and Player 2, you must use the list **append** method.)

Example code:

```
new_game = new_game_positions(1)
print(new_game)
new_game = new_game_positions(5)
print(new_game)
new_game = new_game_positions(7)
print(new_game)
```

Output:

```
[0, 1, 1, 1, 1, 2, 2, 2, 2]
[0, 2, 2, 2, 2, 1, 1, 1, 1]
[0, 1, 1, 2, 2, 2, 2, 1, 1]
```

<-----*----->

### Q8) get_perepere. (*Accessing an element in a list.*)

Requirement: When reprinting the game board on the screen after a user moves, a function is required to retrieve the text to be displayed at a particular board position to show the user which piece is there (or else whether that position contains no player piece).

Assignment Task: Complete the function **get_perepere(position, position_list)** that returns a string with a length of three characters that is used to display which player is at a specified position. The function takes two arguments. The argument **position** is an integer that contains a value between 0 to 8 inclusive, specifying the position that the string will be generated for. The second argument **position_list** is a list object containing integer values of either 0, 1 or 2, indicating an empty position, that player 1 is in that position, or that player 2 is in that position, respectively.

You will need to call the **get_perepere_char()** function that takes a single integer parameter – 0 for an empty position, 1 for player 1 and 2 for player 2, and returns a string representing the equivalent perepere symbol on the game board. If the parameter is 0, the **get_perepere()** function should return the string obtained from the **get_perepere_char()** function surrounded by a pair of square brackets. Otherwise the **get_perepere()** function should return the string obtained from the **get_perepere_char()** function surrounded by a space on either side.

(Note: the function **get_perepere_char(perepere_type)** is provided for your reference here and in the skeleton program. You can assume that this function is already implemented in CodeRunner.

```
def get_perepere_char(perepere_type):
    if perepere_type == 1:
        perepere_char = "֍" # Player 1 (computer)
    elif perepere_type == 2:
        perepere_char = "o" # Player 2 (user)
    else:
        perepere_char = " " # Empty position
    return perepere_char
)
```

Example code:

```
position_list = [0,1,1,1,1,2,2,2,2]
print(get_perepere(0,position_list))
print(get_perepere(1,position_list))
print(get_perepere(5,position_list))
```

Output:

```
[ ]
 ֍
 o
```

<-----*----->

### Q9) replay_game. (Loop through elements in a list)

Requirement: Allow the user to replay a finished game move-by-move.

Assignment Task: Complete the function **replay_game(pos_list, move_list)** that prints each move by each player. The function takes the integer list **pos_list** that specifies the starting positions on the game board (papa tākaro). **pos_list** has nine elements, each element representing a position on the

board (0 through 8), and each element containing an integer value of 0, 1, or 2 depending on whether the position is empty (0), contains a player 1 perepere, or contains a player 2 perepere. The integer list **move_list** specifies the board position that each player moves their piece *from*—starting with player 1's first move and then alternating with player 2's next move, then player 1's next move, and so on.

You must loop through the **move_list** list, updating **pos_list** with the latest positions of perepere on the board. For each move, you must print a string of the form "`>> Player 1 moves from 6 to 0`", correctly displaying the player number, the position that the player is moving from, and the position that the player is moving to. You must use the **draw_papa_taakaro(pos_list)** function to display the game move. After displaying the first move, you must prompt the user to "Press ENTER to continue." before continuing to any subsequent move.

(Note: the **draw_papa_taakaro()** function is supplied in the skeleton program. For another example of a game replay, see the "Example output" section later in this document.)

Example code:

```
pos_list = [0,1,1,1,1,2,2,2,2]
move_list = [1,8]
replay_game(pos_list, move_list)
```

Output (with user hitting the ENTER key marked in magenta):

```
                3
                ֍
        2       .       4
        ֍ ____./ \.____ ֍
          '.           .'
     1  _.-'    0    '-._  5
     ֍   '-.   [ ]  .-'    o
          .'         '.
        8 ----'\ /'---- 6
        o'              o
                7
                o

Replaying game:

>> Player 1 moves from 1 to 0

                3
                ֍
        2       .       4
        ֍ ____./ \.____ ֍
          '.           .'
     1  _.-'    0    '-._  5
    [ ]  '-.    ֍   .-'    o
          .'         '.
        8 ----'\ /'---- 6
        o'              o
                7
                o
Press ENTER to continue.[ENTER]
>> Player 2 moves from 8 to 1

                3
                ֍
        2       .       4
        ֍ ____./ \.____ ֍
          '.           .'
```

```
      1  _.-'    0    '-._  5
      o'    -.   ֍.    -'   o
          .'         '.
         8 ----'\ /'---- 6
        [ ]      '       o
                 7
                 o
Press ENTER to continue.[ENTER]
End of game replay.
```

<-----*----->

### Q10) get_moves_to_keewai. (loop through elements in a list)

Requirement: Assuming that a kēwai position (outer point) is empty, a function is required to get a list of possible positions from which the current player might move one of their perepere (playing pieces) to that empty position. The player could possibly move to that position from the pūtahi (centre position) if they have a perepere there, or from either side of the vacant kēwai if they have perepere in either of those positions.

Assignment Task: Complete the function **get_moves_to_keewai(pos_list, empty_point, player, pos_options)** that returns an updated list of string elements, **pos_options**, with each string element appended by this function representing a possible valid move to the empty kēwai (an outer position).

The integer list **pos_list** contains nine elements. The index of each element corresponds to the position on the board. The value of each element in **pos_list** represents the current state of that position on the board. Valid values are 0, 1, or 2 depending on whether the position is empty (0), contains a player 1 perepere, or contains a player 2 perepere.

The integer **empty_point** contains the index of the element in **pos_list** that has a value of 0 (i.e., that point is empty). The variable **player** contains an integer value representing the current player. There are two possible values: the integers 1 or 2 (representing player 1 or player 2, respectively).

- If the **pos_list** element with index 0 equals the **player** then the function must append a string element to **pos_options**. The format of this two-character string element is zero ("0") concatenated with the **empty_point**.
- Check both of the "kēwai" elements in the positions adjacent to **empty_point**. (Both will have an index value in the range 1 to 8. Also 1 and 8 are adjacent.). For either of the adjacent elements, if the value of that integer element is equal to **player** then this is a potentially valid option and the function must append another string element to **pos_options**. The format of this two-character string element is the index value of that adjacent position concatenated with the **empty_point**.

Example code:

```
pos_list = [2,0,1,1,1,2,2,2,1]
spare_point = 1
player = 1
pos_options = ["00"]
pos_options = get_moves_to_keewai(pos_list, spare_point, player, pos_options)
print(pos_options)
```

Output:

```
['00', '81', '21']
```

<-----*----->

### Q11) get_moves_to_puutahi. (loop through elements in a list)

Requirement: Assuming that the pūtahi (central position) is empty, a function is required to get a list of possible positions from which the current player might move one of their perepere (playing pieces) from a kēwai (outer position) to the pūtahi.

Assignment Task: Complete the function **get_moves_to_puutahi(pos_list, player, pos_options)** that returns an updated list of string elements, **pos_options**, with each string element appended by this function representing a possible valid move from a kēwai (an outer position) to the pūtahi (central board position).

The function must check the elements in the integer list **pos_list** from index values of 1 to 8 (the kēwai positions) to check which player's piece is in that position. The variable **player** contains an integer value representing the current player. There are two possible values: the integers 1 or 2 (representing player 1 or player 2, respectively).

For each element in **pos_list**, if the element value matches the **player** value then the function needs to check whether there is an opponent's piece in either of the kēwai adjacent to that piece. For example, if the **player** is 2, and **pos_list** index 3 is 2, then the function needs to check whether either **pos_list** index 2 or **pos_list** index 4 contains a value of 1, which represents an opposition player piece in an adjacent position. (Also, note that positions 1 and 8 are adjacent on the game board.)

If an element in **pos_list** matches the **player** value and there is an opponent's piece in an adjacent position, then this represents a valid move and the function must append a string element to **pos_options**. The format of this two-character string element is the index value of the current **pos_list** element concatenated with a zero ("0").

Example code:

```
pos_list = [0,1,1,1,1,2,2,2,2]
player = 1
pos_options = ["00"]
pos_options = get_moves_to_puutahi(pos_list, player, pos_options)
print(pos_options)
```

Output:

```
['00', '10', '40']
```

<-----*----->

## Assignment 3 Section B (8 marks)

Once you have completed the 11 functions from Section A, download the Assignment 3 skeleton program from the CompSci 101 assignments website:

https://www.cs.auckland.ac.nz/courses/compsci101s2c/assignments/

Rename the file: "YourUsernameA3.py", e.g., dwil237A3.py. Add your 11 functions from Section A to the program and check that the program executes correctly. Submit your completed Python program using the Assignment Dropbox:

https://adb.auckland.ac.nz/Home/

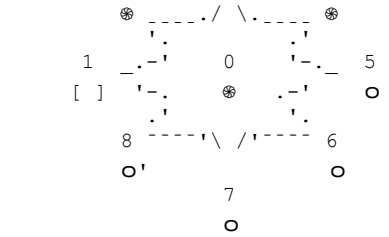
IMPORTANT: Your program MUST include a docstring at the top of the file (containing your name, your username and a correct description of the program) and your program MUST be named correctly (i.e. as stated above).

## Example output

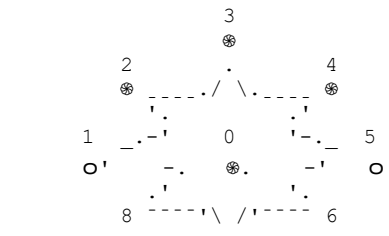
Below is some output produced by the completed program. Your program must execute in the way described and the output should have the same format as the output below. The player input is shown in a bold magenta coloured font.

```
                          -----Mū Tōrere-----
Instructions:
The aim of the game is to move your perepere (play pieces) into a configuration
that blocks your opponent from moving.
Each player has four perepere. Player 1 (Computer) is ֍ and Player 2 is o.
There are nine positions on the board. The the pūtahi (position 0) is in the
centre of the board. This is surrounded by eight kēwai positions (1-8).
- You may move to an empty connected position. You cannot jump another perepere.
- If you are in a kēwai position (1-8), you can move to an adjacent number in
that range if it is unoccupied. (Note that 1 and 8 are adjacent.)
- If you are in a kēwai position (1-8) and the flanking kēwai DO NOT BOTH
contain perepere from your team, you can move to the pūtahi (position 0)
if it is unoccupied.
- If you are in the pūtahi position (0), you can move to a kēwai position
(1-8) if it is empty.


Player 1 move: 1 to 0

                  3
                  ֍
         2        .        4
         ֍  ____./ \.____  ֍
            '.           .'
      1  _.-'    0     '-._  5
     [ ]  '-.     ֍    .-'     o
            .'         '.
         8 ----'\ /'---- 6
         o'              o
                  7
                  o

Which piece do you want to move (or Q to quit)?: 8

                  3
                  ֍
         2        .        4
         ֍  ____./ \.____  ֍
            '.           .'
      1  _.-'    0     '-._  5
      o'    -.    ֍.     -'    o
            .'         '.
         8 ----'\ /'---- 6
           [ ]      '       o
                  7
                  o

Player 1 move: 0 to 8

                  3
                  ֍
         2        .        4
         ֍  ____./ \.____  ֍
            '.           .'
      1  _.-'    0     '-._  5
      o'    -.    [ ]   .-'    o
            .'         '.
         8 ----'\ /'---- 6
         ֍        '       o
```

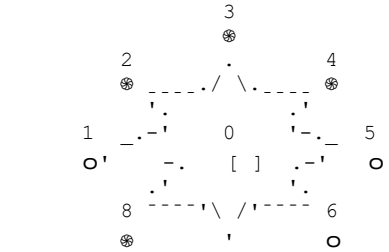
```
                7
                o

Which piece do you want to move (or Q to quit)?: 8

Invalid move! Kia tūpato!

Which piece do you want to move (or Q to quit)?: 1

                3
                ֍
          2     .     4
          ֍ ____./ \.____ ֍
            '.         .'
       1  _.-'   0   '-._  5
      [ ]  '-.   o'   -.   o
            .'       '.
          8 ----'\ /'---- 6
          ֍      '      o
                 7
                 o

Player 1 move: 8 to 1

                3
                ֍
          2     .     4
          ֍ ____./ \.____ ֍
            '.         .'
       1  _.-'   0   '-._  5
       ֍   '-.   o'   -.   o
            .'       '.
          8 ----'\ /'---- 6
         [ ]     '      o
                 7
                 o

Which piece do you want to move (or Q to quit)?: 7

                3
                ֍
          2     .     4
          ֍ ____./ \.____ ֍
            '.         .'
       1  _.-'   0   '-._  5
       ֍   '-.   o'   -.   o
            .'       '.
          8 ----'\ /'---- 6
          o'             o
                 7
                [ ]

You win! Ka rawe!
Play again (Y or N or R for replay)?:r
                3
                ֍
          2     .     4
          ֍ ____./ \.____ ֍
            '.         .'
       1  _.-'   0   '-._  5
       ֍   '-.  [ ]  .-'   o
            .'       '.
          8 ----'\ /'---- 6
          o'             o
                 7
                 o

Replaying game:

>> Player 1 moves from 1 to 0

                3
                ֍
          2     .     4
```



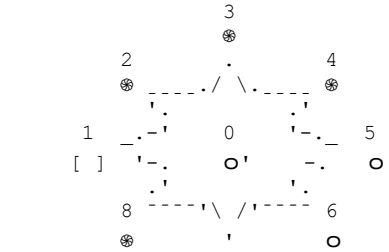

Press ENTER to continue.**[ENTER]**
>> Player 2 moves from 8 to 1



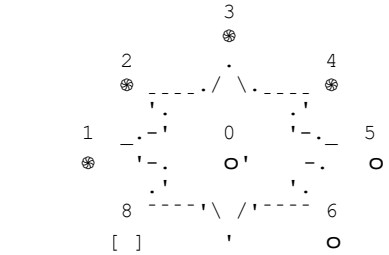

Press ENTER to continue.**[ENTER]**
>> Player 1 moves from 0 to 8



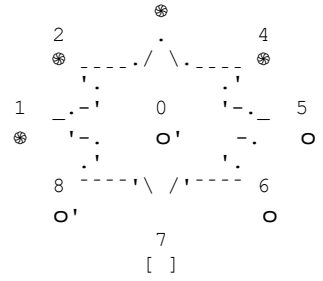

Press ENTER to continue.**[ENTER]**
>> Player 2 moves from 1 to 0



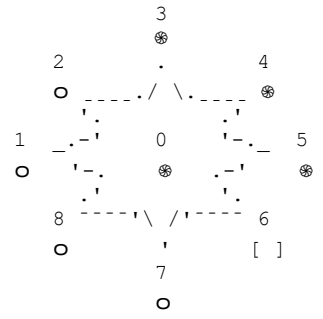

Press ENTER to continue.**[ENTER]**
>> Player 1 moves from 8 to 1



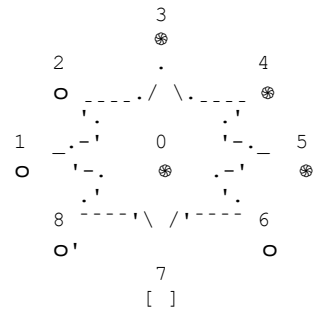

Press ENTER to continue.**[ENTER]**
>> Player 2 moves from 7 to 8



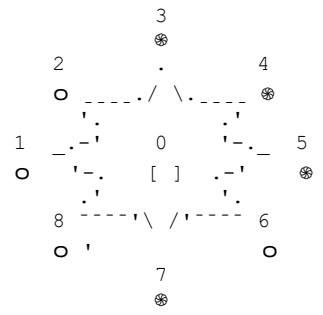

```
Press ENTER to continue.[ENTER]
End of game replay.
Play again (Y or N)?:y

Player 1 move: 6 to 0
```




```
Which piece do you want to move (or Q to quit)?: 7
```




```
Player 1 move: 0 to 7
```




```
Which piece do you want to move (or Q to quit)?: q

Until next time! Mā te wā!
```

## Further Information about Mū Tōrere

A description of Mū Tōrere from Harko Brown [*Ngā Taonga Tākoro* pp. 15-16]:

> **History**
>
> Board games were played prolifically by ancient Māori. The night sky was such a permanent part of their lives, and was a phenomenon that was often studied and discussed. Many games originated from the idea of conceptualising stars and heavenly bodies in game-playing and on papa tākaro (game boards). Mū Tōrere is seen as a combination of the mythology of the wheke (octopus) with eight kāwai (tentacles) simultaneously merged with the revered Matariki and other stars.
>
> Expert players were known to be able to 'see' over forty moves ahead on the eight kēwai boards. The game seemed deceptively simple to early Europeans, but this was a misconception, as there are recordings of groups of settlers being easily beaten by Māori players for large wagers. Today, mū tōrere games are marketed for worldwide sale by several companies. Because of its mathematical properties the game is also used by universities (eg, St Josephs in Philidelphia, U.S.) and other educational institutes around the world.
>
> **Rules**
>
> The object of the game is to move your pieces into positions so as to prevent your opponent from being able to move. The game is played by two players on a papa tākaro, or scribbled into clay or sand on the ground. There are usually eight kēwai/kāwai (points) on a board shaped like an eight-pointed star, although some tribes played with other forty kēwai. However, there is always only one pūtahi. On the eight-point boards each player has four pieces (usually distinctly coloured stones called perepere). Each player gets to start with all their pieces on one half of the board, placed on the four kēwai. The first move always has the starter moving one of their outer pieces into the pūtahi. Each player then moves one piece at a time. Alternatively, if a piece is on one of the kēwai it can be moved into the (empty) central pūtahi or onto one of the (empty) flanking kēwai. Players cannot jump over another piece, or have more than one piece on a kēwai or in the pūtahi at the same time. Players have to move both of their outer perepere before they can move any of their back two perepere.

## Bibliography

**Recommended: (1) Brown, Harko (2008). *Ngā Taonga Tākoro: Māori sports and games*. New Zealand: Penguin Group. (790 B877 General Library Matauranga Maori Level G)**

(2) Ross Calman, 'Traditional Māori games – ngā tākaro - Games of skill with words and hands', *Te Ara - the Encyclopedia of New Zealand*, http://www.TeAra.govt.nz/en/traditional-maori-games-nga-takaro/page-6 (accessed 3 September 2020)

(3) Elsdon Best, *Games and Pastimes of the Maori*. Wellington: Government Printer, 1925, p. 110. Available online at NZETC: http://nzetc.victoria.ac.nz/tm/scholarly/tei-BesGame-t1-body-d5.html