COMP 1039

Problem Solving and Programming

Programming Assignment 1

Tic–Tac–Toe

Prepared by
James Baumeister

UniSA – School of Computer and Information Science

SAIBT Modifications by Andreas Jordan

South Australian Institute of Business and Technology

Contents

INTRODUCTION

This document describes the first assignment for Problem Solving and Programming.

The assignment is intended to provide you with the opportunity to put into practice what you have learnt in the course by applying your knowledge and skills to implement a game of **tic-tac-toe**. It is **very** important that you read and understand this entire document before starting to implement a solution.

This assignment is an **individual task** that will require an **individual submission**. **You will be required to present your work to your practical supervisor during your practical session held in Week 8 of the study period**. Important: You must attend the practical session that you have been attending all study period in order to have your assignment marked.

This document is a kind of specification of the required end product that will be generated by implementing the assignment. Like many specifications, it is written in English and hence will contain some imperfectly specified parts. Please make sure you seek clarification if you are not clear on any aspect of this assignment.

ASSIGNMENT OVERVIEW

You are required to write a Python program that allows a player to play a game of tic-tac-toe against the computer. The program will allow a player to play as many games of tic-tac-toe as they wish.

Tic–Tac–Toe Rules

Tic-tac-toe is a simple paper-based game where:-

- Players mark a nought, 'O', or cross, 'X', on a 3×3 grid.
- A player is either one of the two symbols and the objective of the game is to achieve a vertical, horizontal or diagonal row of three marks.
- The first player to do so is the winner. See https://en.wikipedia.org/wiki/Tic-tac-toe for more information.

It is possible for a game of tic-tac-toe to result in a draw. In fact, two skilled players should always draw! For this assignment, we will implement a simple set of move rules that do not always make the best decisions, so that it is possible for the player to win.

Game Play

- The player chooses the first square in which to place their cross.
- The computer chooses an empty square in which to place its nought.
- This process repeats until either player meets the win condition or it is a draw.
- If either player marks three in a row horizontally, vertically or diagonally the game immediately ends and the winner is declared.
- If all squares are filled, and three in a row has not been achieved, the game is a draw.
- Neither player can change a previous turn, nor create a mark over their opponent's.

PRACTICAL REQUIREMENTS

A graphical user interface (GUI) has been provided for you (see Figure 1). You do not have to write any code that generates the GUI – it should just help you visualise your logic and make the game a little more fun! The GUI is provided as a Python class TicTacToeGUI in a module called tic_tac_toe_gui.py. You will import the module and use the provided methods in your program. Please **do not modify** tic_tac_toe_gui.py.

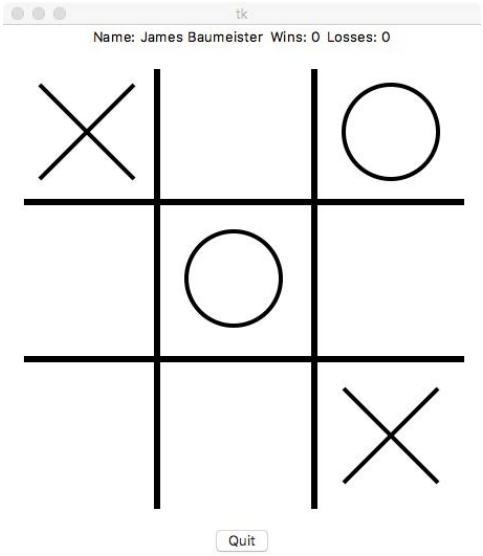

Figure 1: The provided tic-tac-toe GUI

You will make use of the following provided methods:

- `move_computer(slot)`

  This method takes an integer as an argument and draws a nought in the corresponding grid square to represent the computer's move. The slot argument is the list index to be updated. See Figure 3 on Page 8 to see how the tic-tac-toe grid maps to a Python list.

- `increment_wins()`

  This method updates the player's win counter on the GUI. It takes no arguments. It also clears the game board in preparation for the next game.

- `increment_losses()`

  This method updates the player's loss counter on the GUI. It takes no arguments. It also clears the game board in preparation for the next game.

- `draw()`

  This method calls the game a draw. It takes no arguments. It also clears the game board in preparation for the next game.

- `get_wins()`

  This method returns the total number of player wins. It takes no arguments.

Continued on next page ⟶

- `get_losses()`

  This method returns the total number of player losses. It takes no arguments.

The GUI is constructed such that the tic–tac–toe grid is represented by a list of strings, with the string 'X' for the player, 'O' for the computer and empty string `''` for an empty space (see Figure 3).

The provided code mostly just provides the means to interact with the GUI. Your task is to provide a solution for playing a series of tic-tac-toe games against the computer. You will write the basic game rules for the computer's moves, control the flow of play, and display the results to the screen.

It is expected that your solution will include the use of the following:

- Your solution in a file called `your_email_id.py`.

- The supplied tic_tac_toe_gui.py module containing the TicTacToeGui class and methods, as described above. This is provided for you – **please do not modify this file**.

- Appropriate and well constructed while and for loops, as necessary.

- Appropriate `if`, `if-else` and `if-elif-else` statements, as necessary.

- **The following functions:**

  – `display_details()`

  This function takes no arguments and does not return any values. The function displays your details to the screen. Remember that defining the function does not execute the function – you will need to call the function from the appropriate place in the program. Your function should produce the following output (with your details) to the screen:

```
File :  tic-tac-toe.py
Author :  Andreas Jordan
Email Id :  jordanaa
Version :  1.0.5 March 2018
This is my own work as defined by the University's
Academic Misconduct policy.
```

  – `move_computer()`

  This function takes no arguments and does not return any values. It calculates where to move the computer, following these rules in numerical priority:

  1. Check for winning move – if the computer has two noughts in a row and a third space is available, take that space.
  2. Check for the block of a player win – if the player has two crosses in a row and a third space is available, take that space to block.
  3. Check if there are any corners free. If there are, take that space.
  4. Check if the centre is free. If it is, take that space.
  5. Move to any free side space.

  – `check_win(slots, letter)`

  This function takes two arguments: A list of slots representing the grid, and a string holding the letter for which to search for a win condition. The slots parameter should be passed the slots accessible from `tic_tac_toe_gui.py` via `ttt.slots`. The letter should be either 'X' to check for a player win or 'O' to check for a computer win. The function returns a boolean value representing whether the given letter was found

Continued on next page ⟶

to have a win condition. Remember that a win has occurred as soon as a player has three marks in a row, either horizontally, vertically and diagonally. Your solution should search the list passed into the function for any of these occurrences. You can create a copy of the list, but you cannot modify the ttt.slots list in the function.

- `end_game()`

  This function takes no arguments and does not return any values. It should display the end game strings (see Section **SAMPLE OUTPUT** on Page 17) and prompt the user for their choice of whether to continue or not. The only input that should be accepted is a 'y' for yes, or 'n' for no. The program should keep asking if incorrect input is received.

- `display_game(slots)`

  This function takes one argument: A list of slots representing the grid. The slots parameter should be passed the slots accessible from `tic_tac_toe_gui.py` via `ttt.slots`. This function has no return values. Taking Figure 1 as an example, it should print a representation of the grid to the screen in the following format:

```
X |   | O
  | O |
  |   | X
```

- A while loop that allows players to continue playing as many games as they like. A starting while loop has been provided for you:

```
while True:
    try:
        ttt.main_window.update()
    except tkinter.TclError:
        quit(0)
```

  You should not remove or modify any of the starting code. As you add to the while loop, move the try/except block to the bottom. You should also modify the while expression so that it is controlled by the player's play choice.

- Output that strictly adheres to the assignment specifications. If you are not sure about these details, you should check with the 'Sample Output' provided at the end of this document.

- Good programming practice:
  - Consistent commenting, layout and indentation. You are to provide comments to describe: your details, program description, all variable definitions, and every significant section of code.
  - Meaningful variable names.

- **Your solution MUST NOT use:**
  - `quit()`[1] or `exit()` functions
  - `break`, `pass` or `continue` statements as a way to break out of loops. **Doing so will result in a significant mark deduction.**

Your solution should follow these rules exactly **and** make use of the methods provided in `tic_tac_toe_gui.py` to update the GUI.

PLEASE NOTE: You are reminded that you should ensure that all input and output conform to the specifications listed here; if you are not sure about these details you should check with the sample output provided at the end of

[1] The provided code does make use of the `quit()` function, but this is to ensure that our GUI closes cleanly without raising any exceptions.

Continued on next page ⟶

this document or post a message to the discussion forum in order to seek clarification.

Please ensure that you use Python 3.6.2 or a later version (i.e. the latest version) in order to complete your assignments. Your programs MUST run using Python 3.6.2 (or latest version).

STAGES

It is recommended that you develop the assignment in the suggested stages. Many problems in later stages are due to errors in early stages. **Make sure you have finished and thoroughly tested each stage before continuing.**

The following stages of development are recommended:

Stage 1 – Setting Up

You will need the `tic_tac_toe_gui.py` file for this assignment. This has been provided for you. Please download this file from the course website (Assessment tab) and ensure that it is in the same directory as your `your_email_id.py` file.

Test to ensure that this is working correctly by entering the following in your `your_email_id.py` file:

```
import tic_tac_toe_gui
import tkinter

# The object of the TicTacToeGUI class
# that renders the GUI.
# You can use this object to access the
# methods listed in the specification .

ttt = tic_tac_toe_gui.TicTacToeGUI('<name as input>')

# Main game loop.
while True:

    while True:

        # Add your game loop code here.

        # Updates the GUI. DO NOT REMOVE OR MODIFY!
        try:
            ttt.main_window.update()
        except tkinter.TclError, KeyboardInterrupt:
            quit(0)
```

Run the `your_email_id.py` file. If this is working correctly, you should see the window shown in Figure 2.

The `ttt` variable references an object of the `TicTacToeGUI` class. Through it, using the dot notation, you can access the methods discussed in the **Game Play** Section on Page 1.

This is the minimum required code to get the GUI up and running. You should be able to click in any of the nine grid spaces and you will see the player's 'X' mark display. You will only be able to click in one space as the GUI is waiting for the computer's move to be provided.

Note that the provided while loop is deliberately made to be infinitely `True`. You will later need to modify this according to whether or not the player wants to continue playing. You may remove or alter the comments, **but the try/except**

Continued on next page ⟶

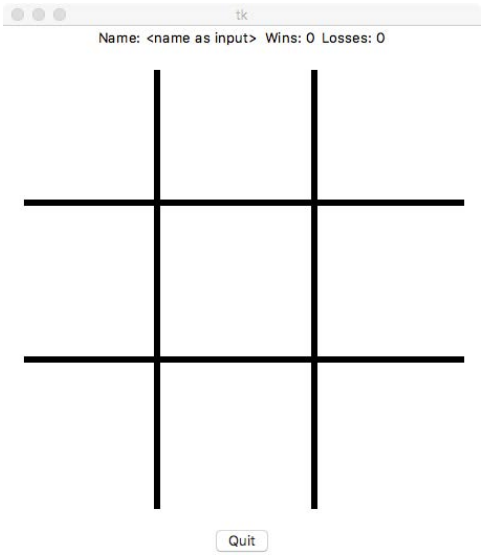

Figure 2: The GUI at Stage 1

block must remain at the bottom of the while suite.

Make sure the program runs correctly. Once you have that working, back up your program. Note: When developing software, you should always have fixed points in your development where you know your software is bug free and runs correctly. Consider teaching yourself a version control system, like Git or SVN.

Stage 2 – Implement Function `display_details()`

key Python concepts: `print` statement

Implement your `display_details()` function, following the output shown below. Think about where the function should be declared in your file in accordance with the Python style guidelines. Also, if you only want it to print once and at the beginning of your program's execution, think about where you should call the function.

```
File         :  tic_tac_toe.py
Author       :  Andy Jordan
Student ID   :  123456789
Email ID     :  joraa001
This is my own work as defined by the University's
Academic Misconduct Policy
```

Stage 3 – Allowing Users To Enter Their Own Name

Key Python concepts: variable creation, variable assignment, `input` statement

The title bar of the GUI shows the player's name and current score. Initially, it displays the string '<name as input>'. You should allow the player to provide their own name via **console input** and *use* that input instead of the default string in the following line of code:

```
ttt = tic_tac_toe_gui.TicTacToeGUI('<name as input>')
```


Continued on next page ⟶

Stage 4 – Implement Function `check_win(slots, letter)`

Key Python concepts: `if/elif/else`, `string` comparison, accessing a `list`

It is now time to implement the function called `check_win(slots, letter)`.

This function accepts two parameters:

- `slots` – A list of tic-tac-toe grid spaces
  - When you call this function, you should pass in the list of strings accessed through the `ttt` variable: `ttt.slots`.
- `letter` – A string representing either of the players: either an 'X' for the human player, or 'O' for the computer.

This function should be written as generically as possible so that:

- it just searches a given list for a series of three in a row of the letter provided (either the 'X' or 'O').

Remember:

- To check for **every possible** win condition – horizontal, vertical and diagonal rows of three.
- Remember to closely follow how the grid spaces map into list indices. The way the grid is mapped to the list is shown in Figure 3 below.
- **If a win condition is found**, this function **should return** `True`. **Otherwise, return** `False`.
- You should also update the GUI by calling `ttt.increment_wins()` if the player won, and `ttt.increment_losses()` if the computer won.

Try testing this function with some dummy lists. For example:

```
test_slots = [' ', ' ', 'X', ' ', 'O', 'X', ' ', 'O', 'X']
print(check_win(test_slots, 'X'))
```

...should return `True`, as the player has three crosses on the rightmost column.

Once you are certain that your solution is correctly identifying all win conditions, you are ready to move on.

Stage 5 – Implement Function `move_computer(ttt)`

Key Python concepts: variable creation, variable assignment, loops, calling functions of an object

Implement the function called `move_computer()`. This requires that you implement the rules as outlined below (copied from page 3). This function **accepts a single parameter**, `ttt`. Within this function you can now use the `ttt` variable to access the list of index positions inside of the `ttt` object (remember that the variable `ttt` *references* the object `ttt`).

Remember that since the human player always goes first, in other words, when a player clicks on the game board (to place a 'O' in the grid), the list ttt.slots will get updated and you can use this to determine if the player has won. You should do this **before** you call your function `move_computer`.
This function needs you to translate the following rules into Python:

1. Check for winning move
   - if the computer has two noughts in a row and a third space is available, take that space.
2. Check for the block of a player win
   - if the player has two crosses in a row and a third space is available, take that space to block.
3. Check if there are any corners free. If there are, take that space.

Continued on next page ⟶

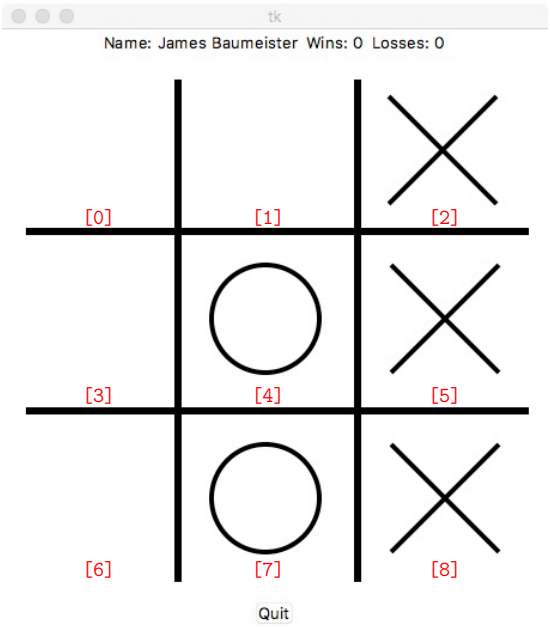

Figure 3: The grid to list index mappings

4. Check if the centre is free. If it is, take that space.
5. If none of the above, then move to any free side space.

Hint – The above steps determine the order in which your code should execute.

In order to do this, you will need access to the list of strings that represents the nine spaces in the tic-tac-toe grid.

You can access the list through the `ttt` variable and saving it to a variable of your own: `slots = list(ttt.slots)`.

- This creates what is known as a 'deep copy'. Anything you do to the `slots` variable in this function will not affect the slots variable in the `TicTacToeGUI` object.

Once your solution has calculated the move, you can update the GUI by calling `ttt.move_computer(slot)`, where slot is the list index that you want updated with a nought ('O').

Hint – The first two rules could make use of the `check_win(slots, letter)` function to see if there is a move that might cause a win.

Stage 6 – Implement Function `end_game()`

Key Python concepts: `input` statement, validating input, returning a value from a function

Implement the `end_game()` function.

- This function takes no arguments and returns a string.
- It should display the end strings to the screen as below (for further examples see Section **SAMPLE OUTPUT** on Page 17)
- Should ask the player if they would like to play again.

You should **validate their input** (especially if you do not want to lose marks!) to make sure that they only enter 'y' for a yes response and 'n' for a no response.

The returned string should be either 'y' or 'n', depending on the choice. You should also alter the main game loop to take into account the player's choice.

Hint – To complete this task, you will need to output the relevant information to the console. This function should also handle asking the user if they want to play again. You should add code to validate user input and using a python **`return`** statement to return the users response (i.e. either 'y' or 'n').

```
---------------------- END GAME ----------------------


That was fun!

Play again?  [y/n] y
```

Stage 7 – Implement Function `display_game(slots)`

Key Python concepts: accessing a list, `print` statement

Implement the `display_game(slots)` function. This function takes a list of the grid slots as an argument. You should pass in the list of grid slots, accessed with `ttt.slots`, just like with the previous functions. You should print the list so that it represents the current game state. Sample output is shown below (see Section **SAMPLE OUTPUT** on Page 17 for further examples).

```
O | O | O
  | X | X
  |   | X
```

Once this function is working correctly, you should call it after every game is finished. This ensures that the final game state is logged.

Hint – For this function you need to access each of the different elements of the list. Then it is a matter of using the **`print`** statement to output to the console.

Stage 8 – Writing the Main Program Loop

Key Python concepts: `if/elif/else`, `string` comparison, accessing a `list`, `input` statement, validating input, calling functions, loops, comparison operations

By now you should have completed coding up the functions in the earlier stages. It is now time to write the main program loop. The main program loop should actually contain **two loops** (one *nested* inside the other).

The first (or **outer loop**) should handle:

- Prompting the user whether they would like to play a game of tic-tac-toe.
  - If the user opts to play (i.e., by pressing 'y'), execution will move to the inner loop.
- Validating the user input.
  - That is, checking if the user entered 'y' or 'n'. Any other input should result in the user being prompted again (see sample output below).

```
Would you like to play Tic Tac Toe?  [y/n] a
Would you like to play Tic Tac Toe?  [y/n] 1
Would you like to play Tic Tac Toe?  [y/n] n
```

The second (or **inner loop**) should handle:

- Checking if the player has won
- Checking if the computer has won
- Checking if the board is full

- Calling `move_computer()`
  - You can check if its the computer's turn by checking the instance variable `ttt.player_turn`. If the value is `False` then you know its the computer's turn.
  - Remember though that certain calls to functions from the `TicTacToeGUI` class also change the value of this instance variable! The table below will help you to determine when you need to change the variable.

Function	`ttt.player_turn`
`ttt.move_computer(slot)`	Set to `True`
`ttt.increment_wins()`	Set to `True`
`ttt.increment_losses()`	Set to `True`
`ttt.draw()`	Set to `True`
`ttt.__init__()`	Set to `True`. While you do not have to worry about this function, it is executed (only once) when the ttt object gets created (in your code, this is the line `ttt = tic_tac_toe_gui.TicTacToeGUI('<name as input>')`). This is also where you could change who goes first when the game starts.
`ttt.__update_slots(slot)`	Set to `False`. While you do not have to worry about this function, it is executed when the player clicks on the game board to take their turn.

Hint: If your code is not correctly checking if the board is full, check that you have initialised your variables appropriately for each iteration of the respective loop. That is any variables used in a loop should get re-initialised either at the beginning (or end) of the loop.
The main program loop is responsible for handling:

- Prompting the user if they wish to play a game of tic-tac-toe
- Calling the function display_game(ttt.slots)
- Displaying an appropriate message after each individual game of tic-tac-toe
  - If computer wins – display `---- Computer wins! ----`
  - If player wins – display `---- Player wins! ----`
  - If board is full and neither player has a win – display `---- Draw! ----`

IMPORTANT – You will need to **modify** the nested loops contained in the `your_email_id.py` file when you first download it. Currently they are set to loop forever. For the outer loop, think about what condition you are testing! That is, whether the user wants to play a game (or another game for subsequent repartitions of the loop). For the inner loop, you need to be checking the conditions to determine if the end of the (current) game has been reached. That is, whether either the computer has won, or the player has won or whether the board is full (and still without a winner).

Stage 9 – Display Game Statistics

By this stage, all components of the game should be working correctly. When the player decides that they do not want to play any longer, you should display the final number of wins, losses and draws. See the sample output in Section **SAMPLE OUTPUT** on Page 17 for how to format this.

Hint: There are methods in the `TicTacToeGUI` class that may be useful for you.

Stage 10 – Matching Sample Output

Finally, check the sample output (see section titled 'Sample Output' towards the end of this document) and if necessary, modify your code so that:

- The output produced by your program EXACTLY adheres to the sample output provided.
- Your program behaves as described in these specs and as the sample output provided.

Note on Functions

In this assignment you are required to write your own user-defined functions. You might not realise but you have been calling functions from the beginning of the course. Examples of functions are (print) and (input).

Defining A Function

Defining a function is simple. It requires you to use a special keyword **def**. For example:

```
def sum(num1, num2):
    the_sum = num1 + num2
    return sum
```

In the example above, we have created a function called **sum**.

- The variables num1 and num2 are *local* to the function.
  - This means that they can only be used inside the function. More importantly, these two variables represent values *passed in* to the function. This is useful as the function can execute its code based on the incoming information contained in the variables.
- Apart from the line of code the_sum = num1 + num2, the only other item of importance is the **return sum** statement. The ability to return values is another useful aspect of functions.
  - These values can be any *single* value, however the value can be a list, a boolean value, a string, an integer or even an object!

Writing code inside a function is no different than writing code anywhere else. You have to remember though that Python uses indentation to mark the beginning and end of blocks of code. For example, when you write if/elif/else or loop statements. So this means that any code to go inside your function, must be indented (using the tab key in IDLE) to indicate it is part of the function.
Based on the above sample function above, the key parts to defining your own function then is:

- The **name** of a function
  - Should always start with a lower case character
  - Should be self-descriptive
  - Must not contain any spaces
- The **parameters** of a function
  - Should always start with a lower case character
  - Should be self-descriptive
  - Must not contain any spaces
  - **Note** that if your function does not take any parameters, you still need the brackets
- A **return** statement
  - Although useful, a function does not have to return a value. If you do not include a return statement in your function, the function will always return a special value called None.
  - A value returned by a function can be assigned to a variable that calls the function. For example, using the function **sum** above, to call it, have it sum two numbers and pass the sum back to us, you could use the following code:

```
number1 = int(input("Enter first number"))
number2 = int(input("Enter second number"))
total = sum(number1, number2)
```

Objects and Object-Oriented Programming

The term object-oriented programming refers to treating just about everything in a program as an *object*! Objects are created, or more appropriately *instantiated* from a `Class`. A program may make use of any number of classes.

Tic-Tac-Toe Class

In this assignment you will use an instance of the `TicTacToeGUI` class called `ttt`. The line of code that instantiates the class is provided in the file `your_email_id.py` and is shown below:

```
ttt = tic_tac_toe_gui.TicTacToeGUI('<name as input>')
```

Table 1 shows the **file** name for the class and its ***class*** name `TicTacToeGUI`.

Filename	Class name
`tic_tac_toe_gui.py`	`TicTacToeGUI`

Table 1: TicTacToeGUI class and file name

The `TicTacToeGUI` class contains numerous methods, most of which you will be able to ignore. These are used to draw the various elements of the graphical user interface (i.e. GUI). There are however a few which you will want to make use of. Furthermore there are a number of *instance variables* which you may also find useful.
Table 2 and 3 list the relevant methods and instance variables that you will want to use in this assignment. All other methods can (and should) be ignored (and hence not listed).

Method name	Description
`move_computer(self, slot)`	Draws a O for the computer turn in the given grid square.
`increment_wins(self)`	Updates the win counter by 1. This method also clears the GUI (of O's and X's) and sets the instance variable, `player_turn` to `True`.
`increment_losses(self)`	Updates the loss counter by 1. This method also clears the GUI (of O's and X's) and sets the instance variable, `player_turn` to `True`.
`get_wins(self)`	Returns the number of player wins.
`get_losses(self)`	Returns the number of player losses.
`draw(self)`	Calls the game a draw. This method also clears the GUI (of O's and X's) and sets the instance variable, `player_turn` to `True`.

Table 2: Relevant methods in the TicTacToeGUI class

Instance variable	Description
`player_turn`	This is a `boolean` variable that is either `True` or `False`. You can use this variable to check if it is the player's turn (or the computer's turn). It can also be used to set the current turn to either the player or the computer.

Table 3: Relevant instance variables in the TicTacToeGUI class

Using `dot` Notation

The variable `ttt` references what is called an `Object`. In order to access the various methods and instance variable(s) belonging to the **`object`**, you must use what is referred to as **`dot`** notation. For example, to increment the number of player wins, you would call the method as follows: `ttt.increment_wins()`

SUBMISSION DETAILS

You are required to demonstrate your assignment to your practical supervisor during your week 8 class for marking. The supervisor will mark your work using the marking criteria included in this document.
You MUST attend the practical session that you have been attending all study period in order to have your assignment marked.

You are also required to submit an electronic copy of your program via Moodle.
Assignments submitted to Moodle, but not demonstrated during your allocated practical session, will NOT be marked.
Likewise, assignments that have been demonstrated during the practical session, but have not been submitted via Moodle, will NOT be marked. Assignments are submitted to Moodle in order to check for plagiarism.

All students must follow the submission instructions below:

- Ensure that your files are named correctly (as per instructions outlined in this document).
- Ensure that the following files are included in your submission:
  - `tic_tac_toe_gui.py` – this will not be modified, but you should submit it anyway.
  - `your_email_id.py` – Remember to replace the default filename to your emailID!
- All files that you submit must include the following comments:

```
#
# File :  filename.py ⟵ Replace with the name of your file
# Author :  your name ⟵ Replace with your name
# Email Id :  your email id ⟵ Replace with your email id
# Version :  1.0 5 July 2023 ⟵ Replace with your version and current month/year
# This is my own work as defined by the University's
# Academic Misconduct policy.
#
```

  Assignments that do not contain these details may not be marked.

You must submit your program **at the start of class before** you demonstrate the work to your marker. You will also be required to demonstrate that you have correctly submitted your work to Moodle. Work that has not been correctly submitted to Moodle will not be marked.

It is expected that students will make copies of all assignments and be able to provide these if required.

Extensions and Late Submissions

There will be no extensions/late submissions for this course without one of the following exceptions:

1. A medical certificate is provided that has the timing and duration of the illness and an opinion on how much the student's ability to perform has been compromised by the illness. Please note if this information is not provided the medical certificate WILL NOT BE ACCEPTED. Late assessment items will not be accepted unless a medical certificate is presented to the Course Coordinator. The certificate must be produced as soon as possible and must cover the dates during which the assessment was to be attempted. In the case where you have a valid medical certificate, the due date will be extended by the number of days stated on the certificate up to five working days.
2. A SAIBT counsellor contacts the Course Coordinator on your behalf requesting an extension. Normally you would use this if you have events outside your control adversely affecting your course work.
3. Unexpected work commitments. In this case, you will need to attach a letter from your work supervisor with your application stating the impact on your ability to complete your assessment.
4. Military obligations with proof.

Applications for extensions must be lodged via learnonline before the due date of the assignment.

Note: Equipment failure, loss of data, 'Heavy work commitments' or late starting of the course are not sufficient grounds for an extension.

Academic Misconduct

Students are reminded that they should be aware of the academic misconduct guidelines available from the SAIBT website (see https://www.saibt.sa.edu.au/policies).

Deliberate academic misconduct such as plagiarism is subject to penalties.

Marking Criteria

Please see feedback form at the end of this document. Other possible deductions:

- Programming style: Things to watch for are poor or no commenting, poor variable names, etc.
- Submitted incorrectly: –10 marks if assignment is submitted incorrectly (i.e. not adhering to the specs).

saibt | University of South Australia

Assessment feedback

Problem Solving and Programming (COMP 1039)
Assignment 1 - Weighting: 10% - Due: Week 8

NAME:	MAX MARK	MARK	COMMENT
PRODUCES CORRECT RESULTS (OUTPUT) Would you like to play Tic Tac Toe? [y/n] y Please enter your name: James ---------- START GAME ---------- --- Player wins! --- X | | O O | O | X | X | X ---------- END GAME ---------- That was fun! Play again? [y/n] n You played 1 games! -> Won: 1 -> Lost: 0 -> Drawn: 0 Thanks for playing! :) --- You played 3 games! -> Won: 1 -> Lost: 1 -> Drawn: 1 ---------- START GAME ---------- --- Player wins! --- X | | O O | O | X | X | X ---------- END GAME ---------- ---------- START GAME ---------- --- Computer wins! --- O | O | O | X | X | | X ---------- END GAME ---------- ---------- START GAME ---------- --- Draw! --- O | X | O O | X | X X | O | X ---------- END GAME ----------	**25 marks**		☐ -2 No or incorrect details at top ☐ -2 No or incorrect line spacing ☐ -1 No or incorrect prompt ('y' or 'n') ☐ -1 No start/end game decoration ☐ -1 No/incorrect display player name ☐ -2 No/incorrect display of game board ☐ -1 No or incorrect game board layout ☐ -2 No or incorrect win/loss update ☐ -1 No or incorrect fun message ☐ -2 No or incorrect prompt ('y' or 'n') ☐ -1 No or incorrect played msg ☐ -1 No or incorrect played value ☐ -2 No or incorrect stats values ☐ -1 No or incorrect game stats layout ☐ -1 No or incorrect thanks message ☐ -2 Additional/excess displays ☐ -1 No or incorrect player win message ☐ -1 No or incorrect computer win message ☐ -1 No or incorrect draw message

Note the above "sample output" is used for **marking purposes only** and should NOT be used to determine formatting requirements. Instead you should reference the SAMPLE OUTPUT at the end of this document.

saibt | University of South Australia

Assessment feedback

Problem Solving and Programming (COMP 1039)
Assignment 1 - Weighting: 10% - Due: Week 8

NAME:	MAX MARK	MARK	COMMENT
ADHERES TO SPECIFICATIONS (CODE)			
Use of `tic_tac_toe_gui.py` module			☐ -1 No or incorrect use of module
Use of `move_computer(slot)` method Use of `increment_wins()` method Use of `increment_losses()` method Use of `draw()` method Use of `get_wins()` method Use of `get_losses()` method			☐ -4 No or incorrect use of provided methods (up to -4 marks)
Check for win condition			☐ -2 No or incorrect win check
Check for draw condition			☐ -2 No or incorrect draw check
Control of computer's turn			☐ -2 No or incorrect computer turn
While loop for play again (playing == 'y')			☐ -2 No or incorrect play again while
Appropriate if statements (general)			☐ -2 No or incorrect if statements
Function `display_details()`			☐ -1 Not to specs or -2 not implemented
Function `move_computer()` Call `check_win(slots, letter)`			☐ -1 Not to specs or -4 not implemented
Function `check_win(slots, letter)`			☐ -1 Not to specs or -3 not implemented
Function `end_game()`			☐ -1 Not to specs or -2 not implemented
Function `display_game(slots)`			☐ -1 Not to specs or -3 not implemented
Main while loop: • Makes use of all above methods • Allows players to repeatedly play • Checks for all win conditions			☐ -2 Not to specs or -4 not implemented
Validation of user input			☐ -1 No validation of user input - play
Good loops			☐ -2 For using break/return/exit statements (or similar) to exit loops
STYLE Comments - • your details • program description • all variable definitions • all functions • code Code layout and spacing Meaningful variable names	**5 MARKS**		☐ -2 Insufficient comments ☐ -2 Inconsistent code layout/spacing ☐ -2 Non-descriptive variable names
TOTAL	**30 MARKS**		

SAMPLE OUTPUT

SAMPLE OUTPUT 1

```
File         :  tic_tac_toe.py
Author       :  James Baumeister
Student ID   :  123456789
Email ID     :  smijy001
This is my own work as defined by the University's
Academic Misconduct Policy

Would you like to play Tic Tac Toe?  [y/n] y
Please enter your name:  James Baumeister

---------------------- START GAME ----------------------

--- Player wins!  ---

 X |   | O
 O | O |
 X | X | X

----------------------- END GAME -----------------------


That was fun!

Play again?  [y/n] y
---------------------- START GAME ----------------------

--- Computer wins!  ---

 O | O | O
   | X | X
   |   | X

----------------------- END GAME -----------------------


That was fun!

Play again?  [y/n] y
---------------------- START GAME ----------------------

--- Draw!  ---
 O | X | O
 O | X | X
 X | O | X

----------------------- END GAME -----------------------
```


Continued on next page ⟶

```
That was fun!

Play again?  [y/n] a

Play again?  [y/n] n


You played 3 games!
 -> Won:      1
 -> Lost:     1
 -> Drawn:    1

Thanks for playing!  :)
```

SAMPLE OUTPUT 2

```
File        :  tic_tac_toe.py
Author      :  James Baumeister
Student ID  :  123456789
Email ID    :  smijy001
This is my own work as defined by the University's
Academic Misconduct Policy

Would you like to play Tic Tac Toe?  [y/n] a
Would you like to play Tic Tac Toe?  [y/n] 1
Would you like to play Tic Tac Toe?  [y/n] n
```