

Assignment 1, Part A: “Not Connect Four”

(20%, due 11:59pm Sunday, April 12th, end of Week 7)

Overview

This is the first part of a two-part assignment. This part is worth 20% of your final grade for IFB104. Part B will be worth a further 5%. Part B is intended as a last-minute extension to the assignment, thereby testing the maintainability of your code, and the instructions for completing it will not be released until Week 7. Whether or not you complete Part B you will submit only one file, and receive only one assessment, for the whole 25% assignment.

Motivation

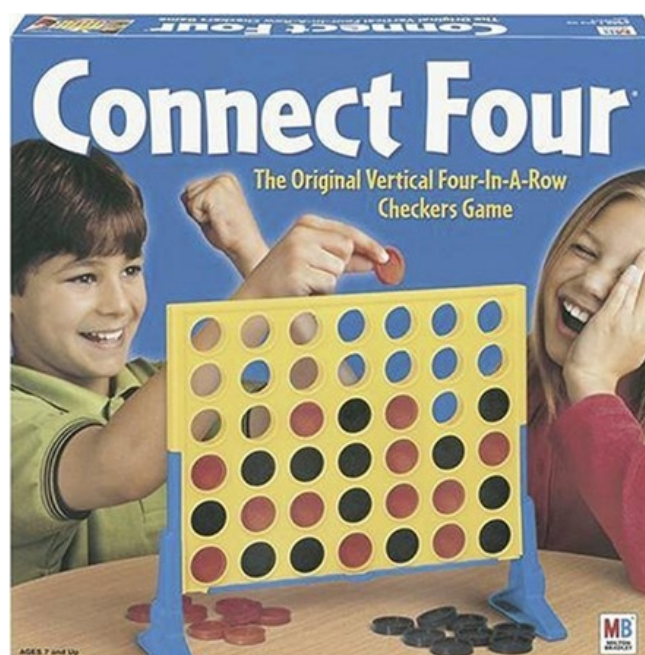
One of the basic functions of any IT system is to process a given data set to produce some form of human-readable output. This assignment requires you to produce a visual image by following instructions stored in a list. It tests your abilities to:

- Process lists of data values;
- Produce maintainable, reusable code;
- Design a general solution to a non-trivial computational problem; and
- Display information in an attractive visual form.

In particular, you will need to think carefully about how to design reusable code segments, via well-planned function definitions and the use of repetition, to make the resulting program as concise as possible and easy to understand and maintain.

Goal

Connect Four (also known as Connect-4 and by many other names) is a popular children’s game in which players take turns dropping coloured tokens into a vertical game board. The first player to complete a vertical, horizontal or diagonal row of four tokens is the winner.



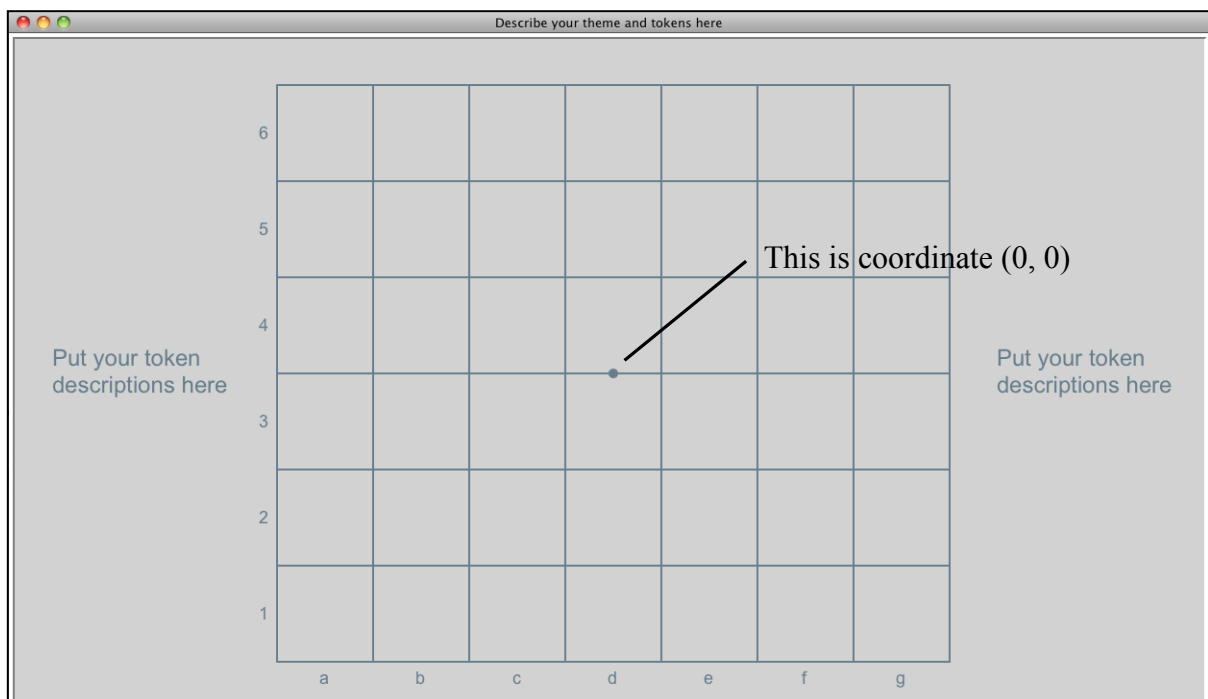
In this assignment you will visually simulate a similar game, but with a number of differences. Our “Not Connect Four” game has:

- four players instead of two;
- random choices of which player gets to drop the next token;
- illustrated square tokens instead of plain round ones; and
- a different way of choosing the winner, which will be revealed only in Part B of the assignment.

To draw the game you must follow a set of instructions, provided as a list of moves, to place the tokens in various cells of the playing board. The tokens must be stacked in columns as in the real game. Most importantly, the sequence of moves to be followed is generated *randomly*, so your solution must be sufficiently general that it can work correctly for *any possible sequence* of moves.

Resources provided

A template Python 3 program, `not_connect_4.py`, is provided with these instructions. When run it creates a drawing canvas and displays a simple image of the playing board on which you will draw specific types of tokens in specific columns. You have a free choice in the design of the four token types, but they must all be related by a common theme. The default image drawn by running the provided Python template appears as shown below. It consists of a numbered grid representing the playing board, with spaces on either side for descriptions of the tokens you have designed.



For convenience, the “home” coordinate (0, 0) is marked by a dot. The board is a 7×6 grid of square cells. Each of the board’s cells measures 100×100 pixels. The spaces to the left and right are where the descriptions of each of your token types will be drawn.

Your task is to extend this template file so that it can draw games of “Not Connect Four” by following a provided list of moves. To do so you must design four entirely distinct tokens, each of which can be drawn in any square on the board. Your code will consist of a function called `play_game` and any auxiliary functions you define to support it. This function takes a single argument, which is a list of moves specifying in which column to place each type of token. The moves are created by a provided function called `random_game` which *randomly* generates the sequence of moves, so your code must work correctly for *any possible* game that could be played!

Designing the tokens

To complete this assignment you must design four *entirely distinct* types of game tokens. Each token must fit exactly into a 100×100 pixel square. Each token must contain a single image, different from the other tokens, and which fills its whole area. They must be drawn using Turtle graphics primitives only, must be easily recognisable, must be of a reasonable degree of complexity involving multiple shapes, and must all be part of some common theme. (Simply drawing the same image in different colours or with some other trivial difference is not considered acceptable.)

You have a free choice of theme and are encouraged to be imaginative! Some possible themes you may consider are:

- Geographical sites (cities, countries or tourist attractions)
- Vehicles (cars, boats, planes, etc)
- Household objects
- Cartoon or comic characters
- TV or movie characters
- Sporting teams
- Businesses (banks, restaurants, IT companies, etc)
- Computer or board games (e.g., Monopoly tokens)
- Internet or cloud service providers
- Or anything else suitable for creating four distinct, related and easily-identifiable tokens

Data format

The `random_game` function used to assess your solution returns a list of moves each representing the action of a player dropping one of their tokens into the board. Each of the moves is expressed as a pair of values with the following general form.

`[column, token_type]`

The column values are letters ranging from ‘a’ to ‘g’ and the token types are integers ranging from 1 to 4. For instance,

`['c', 2]`

tells us to draw a token of type 2 in column 'c'. Note, however, that we are not told in which row to draw the token! As in the real Connect Four game, this depends on how many tokens have already been dropped into this column, if any. Tokens cannot be drawn on top of one another. For instance, if there have already been two tokens placed in column 'c' then this new token must be drawn in row 3.

The `random_game` function generates a list of anywhere between zero and 42 (i.e., 7×6) moves. When choosing which player makes the next move, no attempt is made by the function to be "fair". The next player is chosen at random; we assume some other mechanism, such as rolling a die, is used to choose who moves next. However, the game generated will always be a valid one within the rules of "Not Connect Four". In particular, the moves generated will never overfill a column.

In addition to the `random_game` function, the template file also contains a number of "fixed" data sets. These are provided to help you develop your code, so that you can work with a known sequence of moves, rather than a random one, while debugging your code. However, these "fixed" patterns will not be used for assessing your solution. Your `play_game` function must work correctly for *any* sequence of moves randomly generated by function `random_game`.

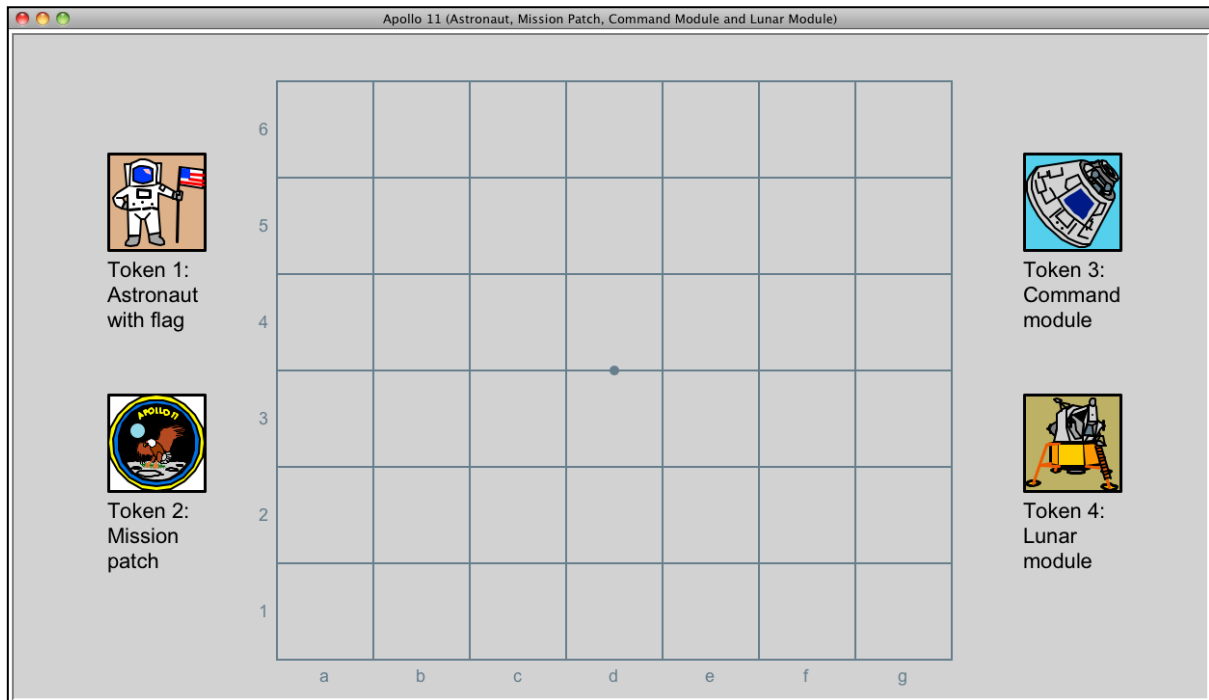
Illustrative example

To illustrate the requirements we developed a solution whose four tokens follow the theme "Apollo 11", in recognition of last year's 50th anniversary of the first moon landing. (Don't copy our example! Come up with your own idea!) We wrote Turtle graphics code that can draw the following four tokens, each representing an aspect of the historic mission.



Each token is *exactly* 100 pixels wide and high, so they all fit perfectly into one of the game board's cells. All of these images were drawn using basic Turtle graphics drawing steps; no separate image files are used to display the tokens. Your images do not need to be as complicated as these examples, but they must still be recognisable and non-trivial.

The first requirement for the assignment is to clearly identify your four tokens and their common theme. To do so you must put an appropriate title on the drawing window and a description of each token in the spaces indicated to the left and right of the game board. Our sample solution is shown overleaf.



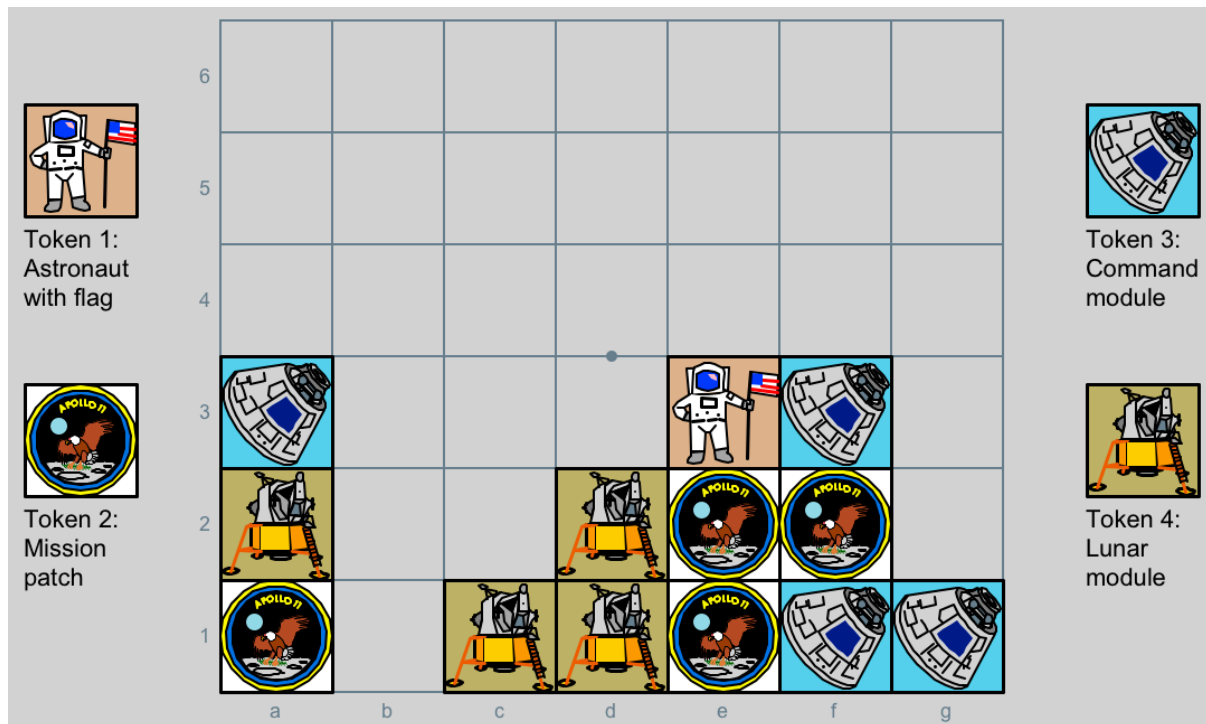
The next requirement is for your implementation of the `random_game` function to draw these tokens in appropriate places on the game board, as per the moves in the list provided as its parameter. To do so you need to take into account the column and token type specified in each move, as well as the number of tokens already placed in that column.

For instance, consider the following list of 13 moves returned by function `random_game`.

```
[[ 'd', 4],
 [ 'e', 2],
 [ 'e', 2],
 [ 'f', 3],
 [ 'a', 2],
 [ 'f', 2],
 [ 'a', 4],
 [ 'c', 4],
 [ 'a', 3],
 [ 'f', 3],
 [ 'e', 1],
 [ 'g', 3],
 [ 'd', 4]]
```

The first move requires us to drop a token of type 4 into column 'd', so our `random_game` function draws the Lunar Module token in row 1 of this column. The next move is a token of type 2 in column 'e', so we draw the Apollo 11 mission patch in row 1 of that column. However, the third move requires us to put another token of the same type in the same column. Since there is already a token in column 'e' we draw the next Lunar Module in row 2. The fourth move is token type 3 in column 'f'. There are no other tokens in this column so

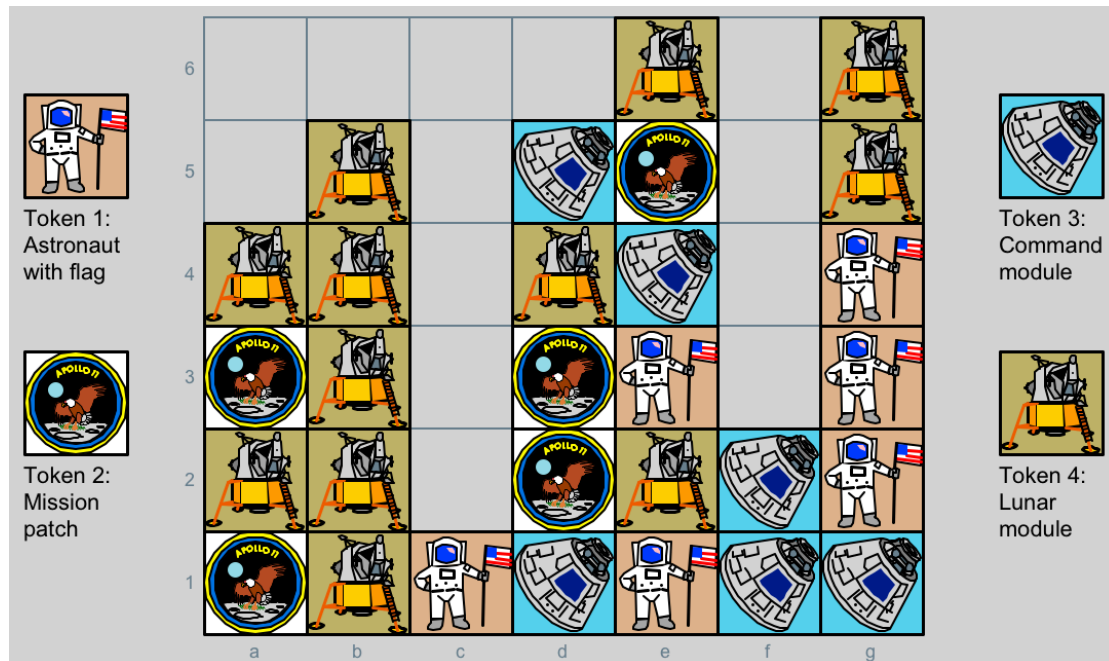
we draw the Command Module in row 1. The fifth move is another token 2 in column 'a', and again this is the first token to appear in that column. The sixth move is token 2 in column 'f' and because this is the second token to appear in that column we draw the mission patch in row 2. This process continues for all the moves in the list, taking care to stack tokens placed in the same column on top of one another. The resulting image, showing all the tokens drawn in this case, is as follows.



Each time we call function `random_game` it produces a different list of drawing instructions. As another example, consider the following longer list returned by the function.

```
[['f', 3], ['e', 1], ['e', 4], ['a', 2], ['b', 4],
 ['d', 3], ['g', 3], ['g', 1], ['a', 4], ['b', 4],
 ['e', 1], ['b', 4], ['a', 2], ['e', 3], ['g', 1],
 ['e', 2], ['f', 3], ['d', 2], ['b', 4], ['d', 2],
 ['g', 1], ['d', 4], ['c', 1], ['d', 3], ['e', 4],
 ['g', 4], ['b', 4], ['a', 4], ['g', 4]]
```

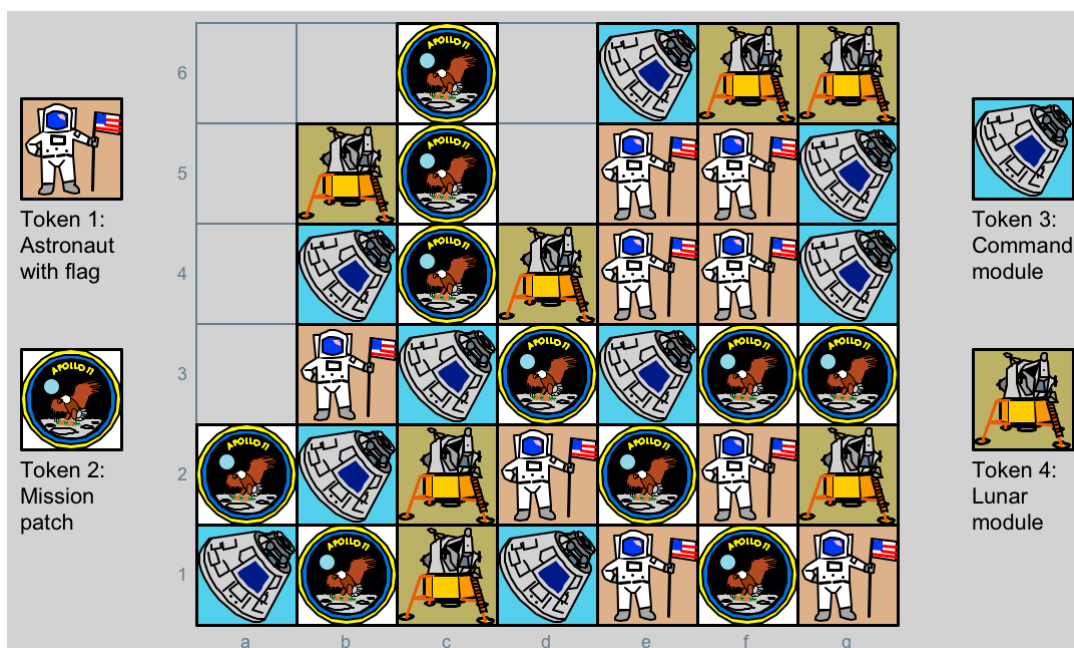
In this case we begin with token 3 in column 'f', followed by tokens of type 1 and 4, both in column 'e', and so on. The resulting image once all 29 moves have been drawn appears below. Notice in the data set above that the only token ever placed in column 'b' is of type 4, which is why we see only Lunar Module tokens in that column.



As an even longer example, consider the following data set of 35 moves.

```
[['f', 2], ['c', 4], ['c', 4], ['e', 1], ['b', 2], ['f', 1],
['g', 1], ['d', 3], ['b', 3], ['b', 1], ['c', 3], ['e', 2],
['f', 2], ['c', 2], ['a', 3], ['a', 2], ['f', 1], ['f', 1],
['g', 4], ['c', 2], ['c', 2], ['d', 1], ['g', 2], ['e', 3],
['f', 4], ['e', 1], ['e', 1], ['b', 3], ['b', 4], ['g', 3],
['d', 2], ['g', 3], ['d', 4], ['e', 3], ['g', 4]]
```

The first move is a token of type 2 in column 'f', then two tokens of type 4 in column 'c', and so on, ending with a type 4 token in column 'g'. The complete game is shown below.



Notice that to produce a nice looking image you can stop the Python template writing “Put your token descriptions here” on the canvas by changing one of the arguments to function `create_drawing_canvas` in the main program. You can also change the background colour and the colour of the lines marking the board in the same way.

Requirements and marking guide

To complete this part of the assignment you are required to extend the provided `not_connect_4.py` Python file by completing function `play_game` so that it can draw a game as specified by the data sets generated by the `random_game` function. Your `play_game` function must work correctly for all possible lists of moves that can be returned by the `random_game` function.

Your submitted solution will consist of a single Python 3 file, and must satisfy the following criteria. Percentage marks available are as shown.

1. **Drawing four entirely distinct tokens within a common theme (5%).** Your program must be able to draw four clearly distinct tokens, each containing a single image different from the other tokens, and each fitting exactly into a 100×100 pixel square. Each token’s image must be clearly recognisable, must be of a reasonable degree of complexity, involving multiple shapes, and must entirely fill the square but without going outside the borders. When drawn on the game board it must be easy to distinguish each token from those adjacent to it through the use of distinct colours and/or borders.
2. **Identifying the theme and tokens (3%).** When executed your code must draw a description of the tokens on the left and right sides of the canvas which clearly indicates the meaning/identity of each of the four tokens. The descriptions must include a visual copy of each token and text describing it. You must also put a title at the top of the Turtle drawing canvas describing your theme and token types.
3. **Placing tokens on the board as per any given list of moves (8%).** Your code must be capable of drawing tokens of the appropriate type in the appropriate columns, exactly as dictated by any valid list of moves provided to function `play_game`. The tokens must be positioned precisely within the cells of the game board. The drawings must preserve their integrity no matter where they are drawn, with no spurious additional or missing lines. Most importantly, tokens must not be drawn on top of one another, but instead must be stacked above any tokens already in the column, if any. Your solution for drawing the tokens must work correctly for *any* lists that can be returned by the `random_game` function.
4. **Code quality and presentation (4%).** Your program code, for both Parts A and B of the assignment, must be presented in a professional manner. See the coding guidelines in the *IFB104 Code Presentation Guide* (on Blackboard under *Assessment*) for suggestions on how to achieve this. In particular, given the obscure and repetitive nature of the code needed to draw complex images using Turtle graphics, each significant code block must be clearly commented to say what it does. Similarly, the names of your functions, parameters and variables should be indicative of their purpose, not just “i”, “j”, etc. Also, you should use function definitions and loops to avoid unnecessary duplication of similar or identical code segments. To get full marks for this

criterion you must provide a significant amount of code to assess; a few lines of well-presented code will not be considered sufficient.

5. **Extra feature (5%).** *Part B of this assignment will require you to make a last-minute change to your solution. The instructions for Part B will not be released until shortly before the final deadline for Assignment 1.*

You must complete the assignment using basic Turtle graphics, random number and maths functions only. You may not import any additional modules or files into your program other than those already included in the given `not_connect_4.py` template. In particular, you may not import any image files for use in creating your drawings.

Finally, you are *not* required to copy the example shown in this document. Instead you are strongly encouraged to be creative in the design of your solution. Surprise us!

Development hints

- Before creating code to draw the individual tokens give careful thought to how you can write the code so that they can be drawn in any place on the canvas. In particular, you need to avoid “hardwiring” your drawing to fixed coordinates.
- The easiest part of this assignment is the description of the tokens and the theme, because these elements never change, so you may want to complete this part first.
- The columns are labelled using letters instead of numbers in this game, so it can be awkward to convert them into coordinates on the Turtle graphics canvas. An elegant solution can be achieved by noting that each letter of the alphabet is actually represented by an “ordinal” number in your computer. In Python you can convert from numbers to letters using the built-in function `chr`, and from letters to numbers using built-in function `ord`. For instance, `chr(78)` returns character 'N', whereas `ord('A')` returns integer 65 and `ord('Z')` returns 90. Thus, although you don't need to use it to complete the assignment, the `ord` function can be helpful when converting the column letters into y-axis coordinates expressed in pixels.
- If you are unable to complete the whole task, just submit whatever you can get working. You will receive *partial marks* for incomplete solutions.
- To help you debug your code we have provided some “fixed” patterns. Feel free to use these when developing your program, and add additional ones if you like, but keep in mind that these data sets will not be used for assessing your solution. Your `play_game` function must work for *any* list that can be returned by function `random_game`.
- Part B of this assignment will require you to change your solution in a short space of time. You are therefore encouraged to keep code maintainability in mind while developing your solution to Part A. Make sure your code is neat and well-commented so that you will find it easy to modify when the instructions for Part B are released.

Artistic merit – The Hall of Fame!

You will not be assessed on the artistic merit of your solution, however, a “Hall of Fame” containing the solutions considered the most attractive or ambitious by the assignment mark-

ers will be created on Blackboard. (Sadly, additional marks will not be given to the winners. The only reward is the envy of your peers.)

Portability

An important aspect of software development is to ensure that your solution will work correctly on all computing platforms (or at least as many as possible). For this reason you must **complete the assignment using standard Turtle graphics, random number and maths functions only**. You may *not* import any additional modules or files into your program other than those already imported by the given template file. In particular, you may not import any image files to help create your drawings or use non-standard image processing modules such as `Pillow`.

Security warning and plagiarism notice

This is an individual assessment item. All files submitted will be subjected to software plagiarism analysis using the MoSS system (<http://theory.stanford.edu/~aiken/moss/>). Serious violations of the university's policies regarding plagiarism will be forwarded to the Science and Engineering Faculty's Academic Misconduct Committee for formal prosecution.

As per QUT rules, you are not permitted to copy *or share* solutions to individual assessment items. In serious plagiarism cases SEF's Academic Misconduct Committee prosecutes both the copier and the original author equally. It is your responsibility to keep your solution secure. In particular, **you must not make your solution visible online via cloud-based code development platforms such as GitHub**. Note that free accounts for such platforms are usually public and are thus unsafe. If you wish to use such a resource, do so only if you are certain you have a *private* repository that cannot be seen by anyone else. For instance, university students can apply for a *free* private repository in GitHub to keep their assignments secure (<https://education.github.com/pack>). However, we recommend that the best way to avoid being prosecuted for plagiarism is to keep your work well away from both the Internet and your fellow students!

Deliverable

You must develop your solution by completing and submitting the provided Python 3 file `not_connect_4.py` as follows.

1. Complete the “statement” at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. *We will assume that submissions without a completed statement are not your own work!*
2. Complete your solution by developing Python code to replace the dummy `play_game` function. You must complete your solution using only the standard Python 3 modules already imported by the provided template. In particular, you must *not* use any Python modules that must be downloaded and installed separately because the markers will not have access to these modules. Furthermore, you may *not* import any image files into your solution; the entire game must be drawn using Turtle graphics drawing primitives only, as we did in our sample solution.
3. Submit *a single Python file* containing your solution for marking. Do *not* submit multiple files. Only a single file will be accepted, so you cannot accompany your so-

lution with other files or pre-defined images. **Do not submit any other files! Submit only a single Python 3 file!**

Apart from working correctly your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable, parameter and function names. *Professional presentation* of your code will be taken into account when marking this assignment.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive *partial marks for incomplete solutions*.

How to submit your solution








A link is available on the IFB104 Blackboard site under *Assessment* for uploading your solution file before the deadline (11:59pm Sunday, April 12th, end of Week 7). You can *submit as many drafts of your solution as you like*. You are strongly encouraged to *submit draft solutions* before the deadline as insurance against computer or network problems near the deadline. If you are unsure whether or not you have successfully uploaded your file, upload it again!

Students who encounter problems uploading their Python files to Blackboard should contact *HiQ's Technology Services* (<http://qut.to/ithelp>; askqut@qut.edu.au; 3138 2000) for assistance and advice. Teaching staff will *not* be available to answer email queries on the weekend the assignment is due, and Technology Services offers limited support outside of business hours, so ensure that you have successfully uploaded at least one solution by close-of-business on Thursday, April 9th.
































Appendix: Some standard Turtle graphics colours

To help you draw your images, below are some commonly-used Turtle graphics colours. The named colours available can vary depending on the computing platform. Those below are commonly cited on the Internet, but these names may not all be supported on all platforms.





















Grays



Color Name	RGB CODE	HEX #	Sample
Black	0-0-0	000000	
Dark Slate Gray	49-79-79	2f4f4f	
Dim Gray	105-105-105	696969	
Slate Gray	112-138-144	708090	
Light Slate Gray	119-136-153	778899	
Gray	190-190-190	bebebe	
Light Gray	211-211-211	d3d3d3	

Whites/Pastels

Color Name	RGB CODE	HEX #	Sample
Snow	255-250-250	fffafe	
Snow 2	238-233-233	eee9e9	
Snow 3	205-201-201	cdc9c9	
Snow 4	139-137-137	8b8989	
Ghost White	248-248-255	f8f8ff	
White Smoke	245-245-245	f5f5f5	
Gainsboro	220-220-220	dcdcdc	
Floral White	255-250-240	fffaf0	
Old Lace	253-245-230	fdf5e6	
Linen	240-240-230	faf0e6	
Antique White	250-235-215	faebd7	
Antique White 2	238-223-204	eedfcc	
Antique White 3	205-192-176	cdc0b0	
Antique White 4	139-131-120	8b8378	
Papaya Whip	255-239-213	ffe4d5	
Blanched Almond	255-235-205	ffebcd	
Bisque	255-228-196	ffe4c4	
Bisque 2	238-213-183	eed5b7	
Bisque 3	205-183-158	cdb79e	
Bisque 4	139-125-107	8b7d6b	
Peach Puff	255-218-185	ffdab9	
Peach Puff 2	238-203-173	eecbad	
Peach Puff 3	205-175-149	cdaf95	
Peach Puff 4	139-119-101	8b7765	
Navajo White	255-222-173	ffdead	
Moccasin	255-228-181	ffe4b5	
Cornsilk	255-248-220	fff8dc	
Cornsilk 2	238-232-205	eee8dc	
Cornsilk 3	205-200-177	cdc8b1	
Cornsilk 4	139-136-120	8b8878	
Ivory	255-255-240	fffff0	

Greens

Color Name	RGB CODE	HEX #	Sample
Medium Aquamarine	102-205-170	66cdaa	
Aquamarine	127-255-212	7fffd4	
Dark Green	0-100-0	006400	
Dark Olive Green	85-107-47	556b2f	
Dark Sea Green	143-188-143	8fbc8f	
Sea Green	46-139-87	2e8b57	
Medium Sea Green	60-179-113	3cb371	
Light Sea Green	32-178-170	20b2aa	
Pale Green	152-251-152	98fb98	
Spring Green	0-255-127	00ff7f	
Lawn Green	124-252-0	7cfc00	
Chartreuse	127-255-0	7fff00	
Medium Spring Green	0-250-154	00fa9a	
Green Yellow	173-255-47	adff2f	
Lime Green	50-205-50	32cd32	
Yellow Green	154-205-50	9acd32	
Forest Green	34-139-34	228b22	
Olive Drab	107-142-35	6b8e23	
Dark Khaki	189-183-107	bdb76b	
Khaki	240-230-140	f0e68c	

Ivory 2	238-238-224	eeeeee0	
Ivory 3	205-205-193	cdcdc1	
Ivory 4	139-139-131	8b8b83	
Lemon Chiffon	255-250-205	fffacd	
Seashell	255-245-238	fff5ee	
Seashell 2	238-229-222	eee5de	
Seashell 3	205-197-191	cdc5bf	
Seashell 4	139-134-130	8b8682	
Honeydew	240-255-240	f0ffff	
Honeydew 2	244-238-224	e0eee0	
Honeydew 3	193-205-193	c1cdc1	
Honeydew 4	131-139-131	838b83	
Mint Cream	245-255-250	f5fffa	
Azure	240-255-255	f0ffff	
Alice Blue	240-248-255	f0f8ff	
Lavender	230-230-250	e6e6fa	
Lavender Blush	255-240-245	fff0f5	
Misty Rose	255-228-225	ffe4e1	
White	255-255-255	ffffff	

Blues

Color Name	RGB CODE	HEX #	Sample
Midnight Blue	25-25-112	191970	
Navy	0-0-128	000080	
Cornflower Blue	100-149-237	6495ed	
Dark Slate Blue	72-61-139	483d8b	
Slate Blue	106-90-205	6a5acd	
Medium Slate Blue	123-104-238	7b68ee	
Light Slate Blue	132-112-255	8470ff	
Medium Blue	0-0-205	0000cd	
Royal Blue	65-105-225	4169e1	
Blue	0-0-255	0000ff	
Dodger Blue	30-144-255	1e90ff	
Deep Sky Blue	0-191-255	00bfff	
Sky Blue	135-206-250	87ceeb	
Light Sky Blue	135-206-250	87cefa	
Steel Blue	70-130-180	4682b4	
Light Steel Blue	176-196-222	b0c4de	
Light Blue	173-216-230	add8e6	
Powder Blue	176-224-230	b0e0e6	
Pale Turquoise	175-238-238	afeeee	
Dark Turquoise	0-206-209	00ced1	
Medium Turquoise	72-209-204	48d1cc	
Turquoise	64-224-208	40e0d0	
Cyan	0-255-255	00ffff	
Light Cyan	224-255-255	e0ffff	
Cadet Blue	95-158-160	5f9ea0	

Yellow

Color Name	RGB CODE	HEX #	Sample
Pale Goldenrod	238-232-170	eee8aa	
Light Goldenrod Yellow	250-250-210	fafad2	
Light Yellow	255-255-224	ffffe0	
Yellow	255-255-0	ffff00	
Gold	255-215-0	ffd700	
Light Goldenrod	238-221-130	eedd82	
Goldenrod	218-165-32	daa520	
Dark Goldenrod	184-134-11	b8860b	

Browns

Color Name	RGB CODE	HEX #	Sample
Rosy Brown	188-143-143	bc8f8f	
Indian Red	205-92-92	cd5c5c	
Saddle Brown	139-69-19	8b4513	
Sienna	160-82-45	a0522d	
Peru	205-133-63	cd853f	
Burlywood	222-184-135	deb887	
Beige	245-245-220	f5f5dc	
Wheat	245-222-179	f5deb3	
Sandy Brown	244-164-96	f4a460	
Tan	210-180-140	d2b48c	
Chocolate	210-105-30	d2691e	
Firebrick	178-34-34	b22222	
Brown	165-42-42	a52a2a	

Oranges

Color Name	RGB CODE	HEX #	Sample
Dark Salmon	233-150-122	e9967a	
Salmon	250-128-114	fa8072	
Light Salmon	255-160-122	ffa07a	
Orange	255-165-0	ffa500	
Dark Orange	255-140-0	ff8c00	
Coral	255-127-80	ff7f50	
Light Coral	240-128-128	f08080	
Tomato	255-99-71	ff6347	
Orange Red	255-69-0	ff4500	
Red	255-0-0	ff0000	

Pinks/Violets

Color Name	RGB CODE	HEX #	Sample
Hot Pink	255-105-180	ff69b4	
Deep Pink	255-20-147	ff1493	
Pink	255-192-203	ffc0cb	
Light Pink	255-182-193	ffb6c1	
Pale Violet Red	219-112-147	db7093	
Maroon	176-48-96	b03060	
Medium Violet Red	199-21-133	c71585	
Violet Red	208-32-144	d02090	
Violet	238-130-238	ee82ee	
Plum	221-160-221	dda0dd	
Orchid	218-112-214	da70d6	
Medium Orchid	186-85-211	ba55d3	
Dark Orchid	153-50-204	9932cc	
Dark Violet	148-0-211	9400d3	
Blue Violet	138-43-226	8a2be2	
Purple	160-32-240	a020f0	
Medium Purple	147-112-219	9370db	
Thistle	216-191-216	d8bfd8	