

## Assignment 1, Part A: “Land Grab”

(20%, due 11:59pm Sunday, September 6th, 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

An important application of computers is to model, simulate and visualise real-life phenomena, such as changing weather patterns, the spread of bush fires, the transmission of viruses, changing land usage, and so on. Here you will write a program that visually displays the outcome of such a simulation.



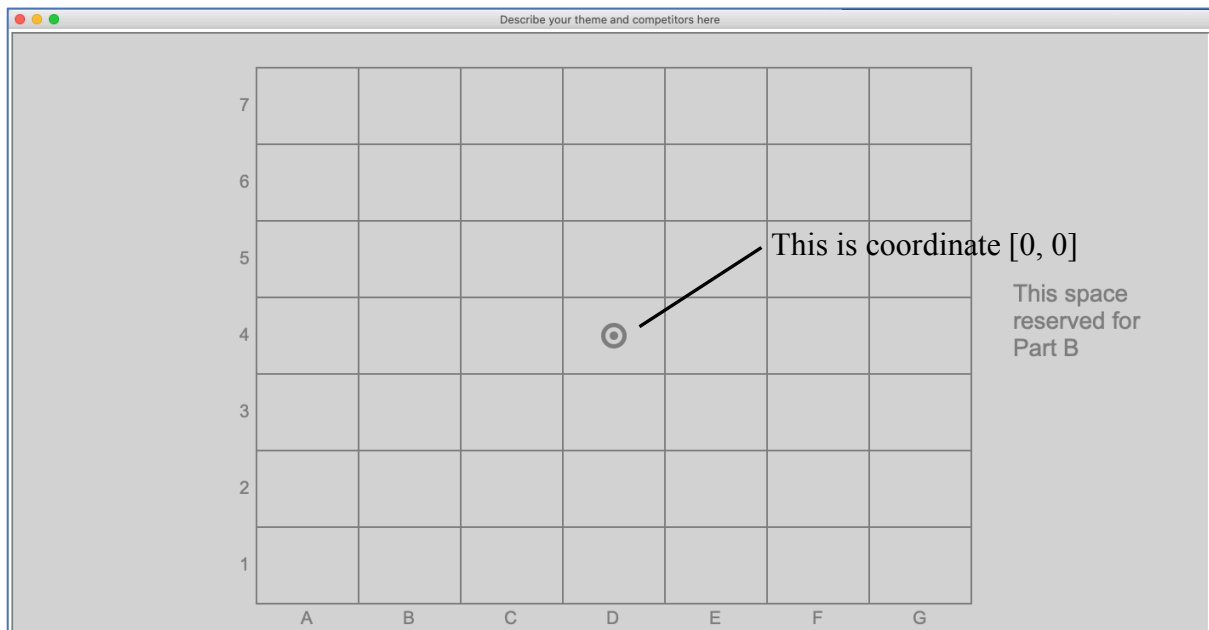
Many real-life systems involve multiple competing forces. For our purposes we assume that four such agents are competing to occupy a limited territory, which could be physical or abstract. Each competing agent moves from its most-recently occupied location in one of four

directions, left, right, up or down. The simulation consists of a sequence of rounds in which each competitor moves once. A competitor can take over a location previously occupied by another agent, but they must not be allowed to move outside the overall territory.

Your code will be given an ordered list of the moves made by the competitors. To visualise the simulation it must display the outcome of each move, so that the user can see how much of the territory each competitor occupies, but without allowing the competitors to go outside the territory. The space available for occupation is represented as a rectangular grid. 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, `land_grab.py`, is provided with these instructions. When executed it creates a drawing canvas and displays a simple image of a grid on which you will draw the simulated competition. The default image drawn by running the provided Python template appears as shown below.



The centre of the middle cell is marked prominently. This is the Turtle graphics “home” coordinate `[0, 0]`. The grid is a  $7 \times 7$  matrix (which gives all competitors the same amount of space to potentially occupy). Each of the grid’s cells measures  $120 \times 90$  pixels (which fits neatly in contemporary “widescreen” computer monitors). There is an empty space to the right which will be used for Part B of the assignment.

After drawing the grid the template prints a sequence of moves to the shell window. Your task is to extend the template file so that it can draw a simulation by following this list of moves. To do so you must design four entirely distinct competitors, each of which can be drawn in any cell in the grid. Your code will consist of a function called `process_moves` and any auxiliary functions you define to support it. This function takes a single argument, which is a list of moves specifying the way in which the competitors move. The moves are created by a provided function called `random_moves` which *randomly* creates the sequence of moves, so your code must work correctly for *any possible sequence* that could be produced!

### ***Designing the competitors***

To complete this assignment you must design four *entirely distinct* competitor images that are linked by a common theme. Each image must fit within a  $120 \times 90$  pixel rectangle, must fill the whole area, and may not go outside of the rectangle. They must be drawn using Turtle graphics primitives only, must be easily recognisable, and must be of a reasonable degree of complexity involving multiple shapes. The images must all be significantly different; simply drawing the same image in different colours or with some other trivial difference is not acceptable.

Apart from these technical constraints you have an entirely free choice of theme and are encouraged to be imaginative! Think of a situation where multiple forces compete with one another to occupy some “territory”, whether physical or abstract. Some examples:

- Natural phenomena (plants, animals, viruses, climatic conditions, geological forces, astronomical forces, etc)
- Artificial phenomena (competing nations, engineering feats, farming or mining claims, etc)
- Conceptual phenomena (philosophies, beliefs, theories, loyalties, etc)
- Fictional characters (detectives and criminals, rival lovers, superheroes and supervillains, adventurers and monsters, etc)
- Competing businesses (banks, restaurants, IT companies, entertainment, etc)
- Video game players or characters
- Sporting teams
- Or anything else suitable for creating four distinct, related and easily-identifiable agents competing for the same goal

### ***Data format***

The `random_moves` function used to assess your solution returns a list of moves for each of the four competitors. The moves occur in rounds such that each competitor gets the same number of opportunities to move (whether successful or not). Each of the moves is expressed as a pair of values with the following general form.

`[competitor_identity, direction]`

The competitor identities range from ‘Competitor A’ to ‘Competitor D’ and the directions are ‘Up’, ‘Down’, ‘Left’ and ‘Right’. For instance,

`['Competitor C', 'Left']`

tells us that Competitor C wants to move one cell to the left of the last cell it occupied. Note, however, that not all such moves can succeed. If a proposed move would cause the competitor to go outside the grid then it must be disallowed and is ignored. Thus the number of cells occupied by a competitor at the end of the simulation may be significantly fewer than the number of moves it attempts. Furthermore, a competitor’s moves may take it back over cells it has occupied previously, in which case there is again no net gain in the territory occupied.

Finally, a move may take a competitor into a cell previously occupied by another competitor. In this case the previous image in the cell is simply overwritten.

The `random_moves` function generates a list of random moves such as the one above, so that the order in which competitors move is also chosen randomly, but each competitor gets the same number of moves in total.

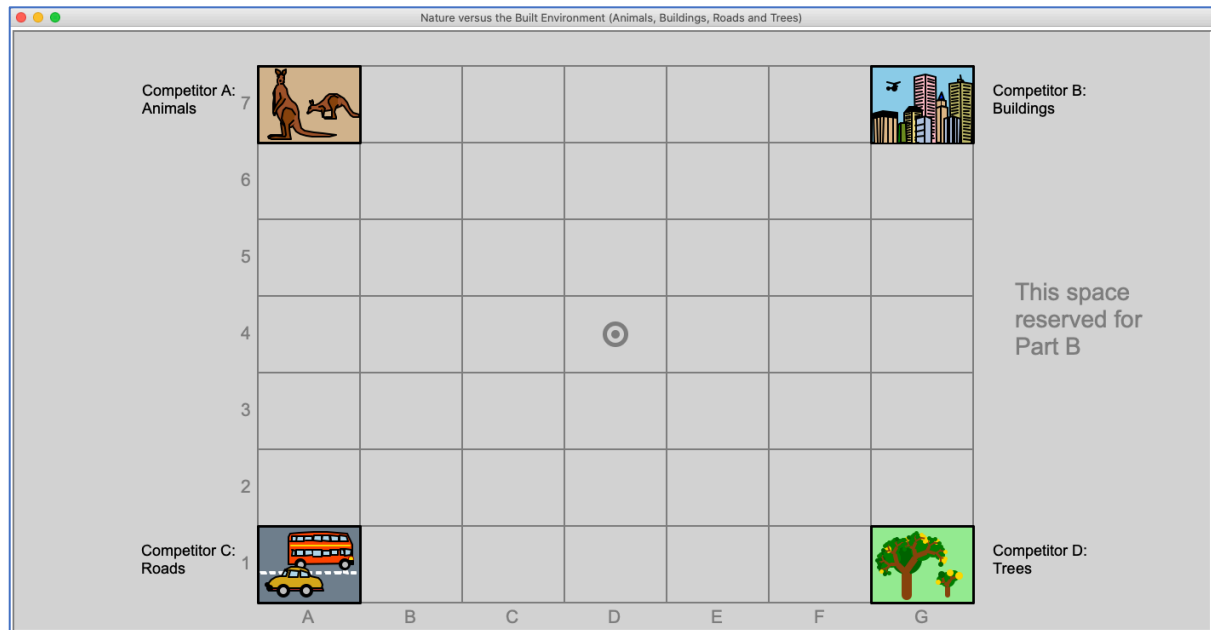
### *Illustrative example*

To illustrate the requirements we developed a solution with four competitors representing the competition for space between the natural and built environments. For the natural environment we chose “animals”, as represented by a mob of kangaroos, and “trees”, as represented by wattle trees with randomly-positioned blossoms. For the built environment we chose “buildings”, depicted as a cityscape, and “roads”, shown by a car and a double-decker bus. (Don’t copy our example! Develop your own idea!) We thus wrote Turtle graphics code that can draw the following four images.



Each image is *exactly* 120 pixels wide and 90 pixels high, so they all fit perfectly into one of the grid’s cells. All of these images were drawn using basic Turtle graphics drawing steps; no separate image files were used. 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 the four competitors and their common theme, and to place them in their starting positions in the grid. Each of the four competitors must start in one corner of the grid. Thus, the initial configuration of our sample solution, before it starts processing the moves, is as shown below. Note the title at the top of the drawing canvas, the correct positioning of the four competitors, and the labels that describe them.



The next requirement is for your implementation of the `process_moves` function to simulate multiple moves of these competing agents by drawing each of their new positions in appropriate cells of the grid, as per the moves in the list provided as the parameter to the function. To do so you will need to keep track of where each competitor currently is.

For instance, consider the following short list of moves returned by function `random_moves`. The list shown below is the one returned by the function call. For convenience during marking and when debugging your code, it is also printed to the shell window.

```
[['Competitor C', 'Right'],
 ['Competitor A', 'Right'],
 ['Competitor B', 'Down'],
 ['Competitor D', 'Left'],
 ['Competitor C', 'Right'],
 ['Competitor A', 'Up'],
 ['Competitor B', 'Down'],
 ['Competitor D', 'Left'],
 ['Competitor C', 'Up'],
 ['Competitor A', 'Left'],
 ['Competitor B', 'Left'],
 ['Competitor D', 'Down'],
 ['Competitor C', 'Up'],
```

```
['Competitor A', 'Down'],
['Competitor B', 'Right'],
['Competitor D', 'Right']]
```

In this case there were four rounds generated, creating 16 moves in total, such that Competitor C moved first, followed by A, B and D in each round. The first move required Competitor C to move to the right, which took it from its home coordinate of (A, 1) in the grid to cell (B, 1). The second move similarly took Competitor A from its home coordinate (A, 7) to (B, 7). The third step moved Competitor B down from (G, 7) to (G, 6), and the fourth move was for Competitor D to go left from (G, 1) to (F, 1), and so on.

Not all moves are successful, however. The sixth move says that Competitor A should go up one place. However, Competitor A is already at the top of the grid in cell (B, 7) and thus cannot go any higher. This move is therefore ignored in our simulation. Another move which is ignored is the 12th in the list, where Competitor D unsuccessfully attempts to move down from cell (E, 1) which is at the bottom of the grid.

Similarly, some successful moves do not result in new territory being occupied. In the tenth move Competitor A goes left and thus returns to its original location (A, 7). In a later move, the 14th in the list, Competitor A moves down to (A, 6) and thus does gain some more territory. Competitor D's final move, to the right, also fails to gain new ground since it goes from cell (F, 5) to a cell it has visited before, (G, 5).

The resulting image, produced by processing all the moves, is as follows in this case. Clearly Competitor C was the most successful at occupying territory in the grid, while Competitors A and D were the least successful.



Of course, function `random_moves` will often produce much longer lists of moves than this. In such cases the competitors may interact with one another by overwriting cells previously occupied by another competitor. As another example, consider the following long list of moves returned by the function. In this case there are 76 distinct moves in 19 rounds. Of



these, 17 moves were ignored because they would have taken a competitor out of the grid, e.g., the first attempted moves by both Competitors D and A were disallowed.

```
[['Competitor B', 'Down'], ['Competitor D', 'Down'], ['Competitor A', 'Up'], ['Competitor C', 'Right'],
['Competitor B', 'Right'], ['Competitor D', 'Up'], ['Competitor A', 'Down'], ['Competitor C', 'Left'],
['Competitor B', 'Right'], ['Competitor D', 'Down'], ['Competitor A', 'Left'], ['Competitor C', 'Left'],
['Competitor B', 'Left'], ['Competitor D', 'Down'], ['Competitor A', 'Down'], ['Competitor C', 'Left'],
['Competitor B', 'Down'], ['Competitor D', 'Up'], ['Competitor A', 'Left'], ['Competitor C', 'Right'],
['Competitor B', 'Left'], ['Competitor D', 'Up'], ['Competitor A', 'Right'], ['Competitor C', 'Right'],
['Competitor B', 'Up'], ['Competitor D', 'Up'], ['Competitor A', 'Up'], ['Competitor C', 'Up'],
['Competitor B', 'Left'], ['Competitor D', 'Down'], ['Competitor A', 'Right'], ['Competitor C', 'Down'],
['Competitor B', 'Down'], ['Competitor D', 'Right'], ['Competitor A', 'Right'], ['Competitor C', 'Right'],
['Competitor B', 'Up'], ['Competitor D', 'Right'], ['Competitor A', 'Left'], ['Competitor C', 'Left'],
['Competitor B', 'Left'], ['Competitor D', 'Down'], ['Competitor A', 'Left'], ['Competitor C', 'Up'],
['Competitor B', 'Up'], ['Competitor D', 'Up'], ['Competitor A', 'Left'], ['Competitor C', 'Up'],
['Competitor B', 'Up'], ['Competitor D', 'Up'], ['Competitor A', 'Down'], ['Competitor C', 'Right'],
['Competitor B', 'Up'], ['Competitor D', 'Right'], ['Competitor A', 'Left'], ['Competitor C', 'Left'],
['Competitor B', 'Down'], ['Competitor D', 'Right'], ['Competitor A', 'Up'], ['Competitor C', 'Right'],
['Competitor B', 'Left'], ['Competitor D', 'Down'], ['Competitor A', 'Right'], ['Competitor C', 'Down'],
['Competitor B', 'Left'], ['Competitor D', 'Left'], ['Competitor A', 'Down'], ['Competitor C', 'Down'],
['Competitor B', 'Left'], ['Competitor D', 'Right'], ['Competitor A', 'Right'], ['Competitor C', 'Left'],
['Competitor B', 'Right'], ['Competitor D', 'Left'], ['Competitor A', 'Down'], ['Competitor C', 'Left']]
```

In this case Competitors A and B clash. Competitor B triumphs, however, by claiming several cells previously occupied by Competitor A and even splits Competitor A's territory in two. (Competitors C and D do not interact with anyone else.) The final result is as follows.<sup>1</sup>



In even longer examples competitors can occupy large parts of the grid, such as the following data set containing 116 moves in 29 rounds, with 32 attempted moves disallowed.

<sup>1</sup> We have never encountered a case where one competitor entirely obliterates another. Can you find one?

```
[['Competitor C', 'Up'], ['Competitor D', 'Left'], ['Competitor B', 'Down'], ['Competitor A', 'Down'],
['Competitor C', 'Left'], ['Competitor D', 'Down'], ['Competitor B', 'Up'], ['Competitor A', 'Down'],
['Competitor C', 'Down'], ['Competitor D', 'Right'], ['Competitor B', 'Left'], ['Competitor A', 'Left'],
['Competitor C', 'Left'], ['Competitor D', 'Down'], ['Competitor B', 'Right'], ['Competitor A', 'Left'],
['Competitor C', 'Right'], ['Competitor D', 'Down'], ['Competitor B', 'Right'], ['Competitor A', 'Down'],
['Competitor C', 'Up'], ['Competitor D', 'Right'], ['Competitor B', 'Down'], ['Competitor A', 'Left'],
['Competitor C', 'Left'], ['Competitor D', 'Left'], ['Competitor B', 'Right'], ['Competitor A', 'Left'],
['Competitor C', 'Right'], ['Competitor D', 'Up'], ['Competitor B', 'Up'], ['Competitor A', 'Left'],
['Competitor C', 'Left'], ['Competitor D', 'Left'], ['Competitor B', 'Left'], ['Competitor A', 'Down'],
['Competitor C', 'Up'], ['Competitor D', 'Up'], ['Competitor B', 'Down'], ['Competitor A', 'Right'],
['Competitor C', 'Right'], ['Competitor D', 'Left'], ['Competitor B', 'Up'], ['Competitor A', 'Left'],
['Competitor C', 'Up'], ['Competitor D', 'Up'], ['Competitor B', 'Right'], ['Competitor A', 'Left'],
['Competitor C', 'Left'], ['Competitor D', 'Up'], ['Competitor B', 'Down'], ['Competitor A', 'Up'],
['Competitor C', 'Down'], ['Competitor D', 'Left'], ['Competitor B', 'Down'], ['Competitor A', 'Down'],
['Competitor C', 'Left'], ['Competitor D', 'Up'], ['Competitor B', 'Right'], ['Competitor A', 'Left'],
['Competitor C', 'Right'], ['Competitor D', 'Up'], ['Competitor B', 'Left'], ['Competitor A', 'Down'],
['Competitor C', 'Down'], ['Competitor D', 'Left'], ['Competitor B', 'Right'], ['Competitor A', 'Left'],
['Competitor C', 'Left'], ['Competitor D', 'Left'], ['Competitor B', 'Right'], ['Competitor A', 'Left'],
['Competitor C', 'Up'], ['Competitor D', 'Left'], ['Competitor B', 'Down'], ['Competitor A', 'Up'],
['Competitor C', 'Down'], ['Competitor D', 'Up'], ['Competitor B', 'Down'], ['Competitor A', 'Left'],
['Competitor C', 'Left'], ['Competitor D', 'Left'], ['Competitor B', 'Right'], ['Competitor A', 'Down'],
['Competitor C', 'Right'], ['Competitor D', 'Down'], ['Competitor B', 'Up'], ['Competitor A', 'Down'],
['Competitor C', 'Up'], ['Competitor D', 'Down'], ['Competitor B', 'Up'], ['Competitor A', 'Left'],
['Competitor C', 'Right'], ['Competitor D', 'Right'], ['Competitor B', 'Right'], ['Competitor A', 'Left'],
['Competitor C', 'Right'], ['Competitor D', 'Up'], ['Competitor B', 'Right'], ['Competitor A', 'Up'],
['Competitor C', 'Right'], ['Competitor D', 'Right'], ['Competitor B', 'Down'], ['Competitor A', 'Down'],
['Competitor C', 'Right'], ['Competitor D', 'Left'], ['Competitor B', 'Down'], ['Competitor A', 'Left'],
['Competitor C', 'Left'], ['Competitor D', 'Left'], ['Competitor B', 'Left'], ['Competitor A', 'Left'],
['Competitor C', 'Up'], ['Competitor D', 'Up'], ['Competitor B', 'Right'], ['Competitor A', 'Right']]
```

The completed simulation for this data set is as shown below.<sup>2</sup> In this case Competitor D succeeds in reaching the opposite corner of the grid although its overall territory was partitioned by Competitor C. Also, Competitor A took over Competitor C's home location.



<sup>2</sup> We have never encountered a case where the entire grid is filled. Can you find one?



To produce a nice looking final image you can stop the Python template writing instructions on the canvas by changing 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 grid in the same way if you want to avoid a conflict with the colour scheme chosen for your competitors.

### ***Requirements and marking guide***

To complete this part of the assignment you are required to extend the provided `land_grab.py` Python file by completing function `process_moves` so that it can draw a simulation as specified by the data sets generated by the `random_moves` function. Your `process_moves` function must work correctly for all possible lists of moves that can be returned by the `random_moves` function when called with no arguments.

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 distinct competitors within a common theme (5%).** Your program must be able to draw four clearly distinct competitors, each represented by a single image clearly different from the others, and each fitting exactly into a  $120 \times 90$  pixel rectangle. Each competitor's image must be clearly recognisable, must be of a reasonable degree of complexity, involving multiple shapes, and must entirely fill the rectangle but without going outside the borders. Each of the four competitor's images must appear in the appropriate corner of the grid when the simulation begins. You must put a title at the top of the Turtle drawing canvas describing your theme and the competitors. Each of the four competitors must also be described on the drawing canvas, in a way that clearly indicates the meaning/identity of each of the four images, and it must be clear which competitor is considered "Competitor A", "Competitor B", etc.
2. **Following the individual moves (7%).** When executed your code must draw a sequence of images for each of the four competitors which follow their 'left', 'right', 'up' and 'down' moves, in the order provided in the given data set. The drawings must preserve their integrity no matter where they are drawn, e.g., with no spurious additional or missing lines. Each move for each competitor must be made relative to its last position in the grid, starting in their initial corner. If necessary, a competitor's image can overwrite that of another competitor (or a previous move by the same competitor). Your solution for drawing the images must work correctly for *any* list that can be returned by the `random_moves` function when called with no arguments. (This criterion only concerns the ability to follow the moves and not the requirement to stay in the grid, which is covered in the next criterion.)
3. **Keeping the competitors in the grid (4%).** Your code must be capable of recognising when a proposed move would cause a competitor to go outside the boundaries of the  $7 \times 7$  grid, and prevent the competitor from doing so by ignoring that move. Only moves that would go outside the grid should be ignored; all other moves in the data set must be followed. Your solution for drawing the images must work correctly for *any* list that can be returned by the `random_moves` function when called with no arguments.
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 guide-

lines 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 describe its *purpose* (but not *how* it works, which can be seen from the code itself). Similarly, the names of your functions, parameters and variables should be indicative of their role, not just “i”, “j”, etc. Also, you should use function definitions and loops to avoid unnecessary duplication of similar or identical code segments. All instructions in the provided Python template must be followed. Finally, 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 `land_grab.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 writing lots of code to draw the individual images give careful thought to how you can ensure they can be drawn at any chosen location in the grid. In particular, you need to avoid “hardwiring” your drawings to fixed coordinates on the canvas.
- The easiest part of this assignment is drawing the competitors in their starting positions, and describing the competitors and the theme, because these elements never change, so you may want to complete this aspect first.
- There are two major computational challenges in the assignment, following the given moves and ensuring that the competitors do not go outside the grid. Rather than trying to solve both problems at once, it’s easiest to do them one after another. First work out how to follow the moves, and only after you’ve done this should you worry about keeping the competitors inside the grid.
- To help you debug your code we have provided a few “fixed” sets of moves in the template file. Similarly, you can force the `random_moves` function to always produce the same data set by giving it a “seed” value as an argument. Feel free to use these features when developing your program, and add additional data sets if you like, but keep in mind that these fixed values will not be used for assessing your solution. Your `process_moves` function must work for *any* list that can be returned by function `random_moves` when called with no arguments.
- If you are unable to complete the whole task, just submit whatever parts you can get working. You will receive *partial marks* for incomplete solutions. However, comment out non-working code so that your program will run when the marker tests it.

- Part B of this assignment will require you to extend 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 markers will be created on Blackboard. (Sadly, additional marks will not be given to the winners. The only reward is watching your peers turn green with envy.)

### ***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!* Remember, when your best friend asks to “just have a look” at your assignment, it means they intend to submit your code with their name on it, landing you both in hot water!

### ***Deliverable***

You must develop your solution by completing and submitting the provided Python 3 file `land_grab.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 `process_moves` 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 solution with other code 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, September 6th, 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](mailto:askqut@qut.edu.au); 3138 2000) for assistance and advice. Teaching staff will *not* be available to answer email queries on the evening the assignment is due, and Technology Services offers limited support outside of business hours, so ensure that you have successfully uploaded at least one draft solution by close-of-business on Friday, September 4th.
































## Appendix: Some standard Turtle graphics colours

To help you draw your images, below are some commonly-cited Turtle graphics colours. The colours available can vary depending on the computing platform. Those below are commonly listed on the Internet, but these names may not 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	ffaafa	
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	dccdc	
Floral White	255-250-240	ffaaf0	
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	ffefd5	
Blanched Almond	255-235-205	ffe4cd	
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	eeebad	
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	ffff0	

### 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	ffffad	
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	f0fff0	
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	

Source: [www.discoveryplayground.com/computer-programming-for-kids/rgb-colors/](http://www.discoveryplayground.com/computer-programming-for-kids/rgb-colors/)