Strathmore UNIVERSITY

MASTER OF SCIENCE IN DATA SCIENCE AND ANALYTICS

HW 3: Tower Blaster

DSA 8101: Fundamental Computing Concepts

HW deadline 12th August 2022.

This homework (HW) deals with the following topics:

- Lists
- tuples

In this HW, we will be implementing the game Tower Blaster, which is a game that involves re-arranging a group of bricks to have an increasing sequence.

About the Game

Tower Blaster is often played with 2 human players, but we will keep this simple and just play the user versus the computer. The user's moves are decided by the user playing the game, by asking for input, and the computer's moves are decided by the program.

There is NO right answer for the section that asks you to program a strategy for the computer. All we want you to do is come up with a reasonable enough strategy that ensures a human user does not always beat the computer. So, unlike your previous assignments, this one has a creative component to it.

You must use Python lists and tuples for this assignment. Some of you might have seen a package called numpy. You are NOT allowed to use numpy at all.

For an online version of Tower Blaster, go here:
https://yaksgames.com/games/tower-blaster/G9991E56CC

Try to get a feel for the game. Our game will be similar, but read our specifications. Most importantly, we won’t have levels. Playing the game while developing your code can be helpful.

A Tower Blaster game starts with a main pile of 60 bricks, each numbered from 1 to 60. Think of the numbers on the bricks as the width of the bricks. The objective is to be the first player to arrange 10 bricks in your own tower from lowest to highest (from the top down), because the tower will be unstable otherwise.

The bricks in the main pile are shuffled at the start and both the user and the computer are dealt 10 bricks from the main pile. As a player receives each brick, they must place it on top of their current tower in the order it is received. Yes, initially your tower is likely to be unstable.

After the first 10 bricks are dealt to the user and the computer, there will be 40 remaining in the main pile. The top brick of the main pile is turned over to begin the discarded brick pile.

On each player’s turn, the player chooses to pick up the top brick from the discard pile or to pick up the top brick from the main pile. The top brick from the discard pile is known. In other words, the discard pile is ‘face up’ and everyone knows how wide the top brick is. The main pile is ‘face down’. Choosing the top brick from the main pile can be risky, because the player does not know what the brick is.

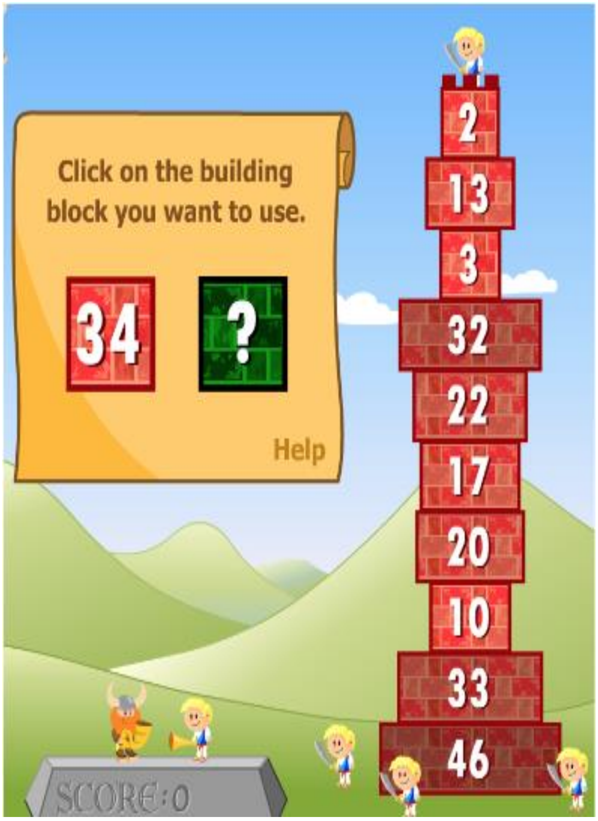
Once a player chooses a brick, either from the discard pile or from the main pile, the player decides where in the tower to put the brick. The tower is always 10 bricks high, so placing a brick means that an existing brick in the tower is removed and replaced with the new brick.

If the player takes a brick from the main pile (the one that is ‘face down’), the player can reject it and place it in the discard pile. This means that nothing in that player’s tower changes during that turn. If the player takes a brick from the discard pile (the one that is ‘face up’), the player MUST place it into the tower.

The first player to get their 10 bricks in order wins.

If, at any point, all of the cards have been removed from the main pile of bricks, then all of the cards in the discard pile are shuffled and moved to the main pile. Then the top card is turned over to start the new discard pile.

The Actual Program



Below you will find explanations of the functions that need to be written. We are expecting to see these functions with these names and method signatures exactly. Do not change the names of these functions, as we will be running automated tests against each individual function.

You will also have to write some functions by yourself, in addition to the ones listed below. Feel free to name those functions whatever you want, as long as they happen to reflect what the function does.

Note that we will make heavy use of lists in this assignment. Since a tower looks more like a vertically oriented structure, we need to have some convention. Our convention is that the tower in the picture to the right is going to be represented as [2, 13, 3, 32, 22, 17, 20, 10, 33, 46]

We know this is a somewhat awkward way to represent the data, but we are deliberately asking you to do it this way in order for you to do more list exercises.

The main pile and discard pile are also going to be represented as lists. Note that in both the main pile and the discard pile, you should only have access to the top.

So, to add/remove bricks, consider the top of the brick pile or tower to be the beginning of the list. You will have to use the pop function, the append function, and the insert function, in some manner.

Required Functions

Be sure to add docstrings to all of your functions and comments to your code.

setup_bricks():

- You’ll run this function once at the beginning of the game.
- Creates a main pile of 60 bricks, represented as a list containing the integers 1 – 60.

- Creates a discard pile of 0 bricks, represented as an empty list.
- This function returns both lists.
- The method of returning 2 things from a function is to make a tuple out of the return values. For an example of this, refer to the lecture slides/code where we return both the maximum and minimum in a list.

shuffle_bricks (bricks):

- Shuffle the given bricks (represented as a list). (You'll do this to start the game.)
- This function does not return anything.
- You are allowed to import the random module and just use random.shuffle.

check_bricks(main_pile, discard):

- Check if there are any cards left in the given main pile of bricks.
- If not, shuffle the discard pile (using the shuffle function) and move those bricks to the main pile.
- Then turn over the top card to be the start of the new discard pile.

check_tower_blaster(tower):

- Given a tower (the user's or the computer's list), determine if stability has been achieved.
- Remember, stability means that the bricks are in ascending order.
- This function returns a boolean value.

get_top_brick(brick_pile):

- Remove and return the top brick from any given pile of bricks. This can be the main_pile, the discard pile, or your tower or the computer's tower. In short, remove and return the first element of any given list.
- It is used at the start of game play for dealing bricks. This function will also be used during each player's turn to take the top brick from either the discarded brick pile or from the main pile.
- Note: Brick piles are vertically oriented structures, with the top having index 0.
- This function must return an integer.

deal_initial_bricks(main_pile):

- Start the game by dealing two sets of 10 bricks each, from the given main_pile.
- Make sure that you follow the normal conventions of dealing. So, you have to deal one brick to the computer, one to the user, one to the computer, one to the user, and so on.
- The computer is always the first person that gets dealt to and always plays first.

- Remember that the rules dictate that you have to place your bricks one on top of the other. In the earlier picture, this would mean that someone was dealt 46, 33, 10, ..., in that order.
- This function returns a tuple containing two lists - one representing the user's hand and the other representing the computer's hand.

add_brick_to_discard(brick, discard):

- Add the given brick (represented as an integer) to the top of the given discard pile (which is a list)
- This function does not return anything.

find_and_replace(new_brick, brick_to_be_replaced, tower, discard):

- Find the given brick to be replaced (represented by an integer) in the given tower and replace it with the given new brick.
- Check and make sure that the given brick to be replaced is truly a brick in the given tower.
- The given brick to be replaced then gets put on top of the given discard pile.
- Return True if the given brick is replaced, otherwise return False.

computer_play(tower, main_pile, discard):

- This function is where you can write the computer's strategy. It is also the function where we are giving you very little guidance in terms of actual code.
- You are supposed to be somewhat creative here, but I do want your code to be deterministic. That is, given a particular tower and a particular brick (either from the discarded brick pile or as the top brick of the main pile), you either always take the brick or always reject it.
- Here are some potential decisions the computer has to make:
    - Given the computer's current tower, do you want to take the top brick from the discarded brick pile or do you want to take the top brick from the main pile and see if that brick is useful to you.
    - How do you evaluate the usefulness of a brick and which position it should go into.
    - There might be some simple rules you can give the computer. For instance, it is disastrous (provably so) to put something like a 5 at the very bottom of the tower. You want big numbers over there.
- You are allowed to do pretty much anything in this function except make a random decision or make a decision that is obviously incorrect. For instance, making your bottom brick a 5 is not very smart and a recipe for disaster.
- Also, the computer CANNOT CHEAT. What does that mean? The computer cannot peek at the top brick of the main pile and then make a decision to go to

the discard pile. Its decisions should be something that a human could be able to make as well.

- This function returns the new tower for the computer.
- Hint: One approach is to try to think of how you, as a human user, would consider selecting a new brick (from the main pile or discard pile) and using it to replace a brick in your own tower.

main():

- The function that puts it all together.
- You play until either the user or the computer gets Tower Blaster, which means their tower stability has been achieved.
- Unlike the previous HW, we want you to assemble the functionality in the main function by yourself. Think about how the game should proceed and where each function should go.
- All user input should take place in the main function. User Interface
- You're free to create your own user interface for the game, as long as it makes sense for the user playing.

The computer goes first. Your program can print the computer's initial tower (in list form) at the beginning of the game, but for the rest of the game, the human player shouldn't see what's in the computer's tower. After the computer's turn, your program should print the results of the turn. For example, your program might print this if the computer chooses the top brick of the discard pile and replaces a brick in their tower.

When it's the human user's turn, your program should print the user's tower (in list form), the top brick of the discard pile, and, if they choose to pick from the main pile, the top brick of the main pile. For example, your program might print this if the human user chooses the top brick of the main pile and replaces a brick in their tower.

Evaluation

The primary goal of this assignment is to get you to feel familiar with lists and tuples and to have some level of fun while creating a game.

While we want you to spend time on coming up with some kind of strategy for the computer, that is NOT the primary part of the assignment. Any simple, but reasonable strategy will have you doing some fun things with lists. If the human player always does absolutely nothing at all, that is, if they reject the top discarded brick and they reject the top brick from the main pile and move it onto the discard, then we want the computer to win. Your computer player should be smart enough to beat the ‘stupid, lazy’ user.

You will be evaluated on 4 things:

1. **Does your game work - 5 points**
   Again, please do not worry about the TAs trying to crash your program. Just ensure that reasonable inputs will allow a user to play the game.

2. **Function design – 5 points**
   In addition to your game working as it should, we will confirm that the required functions above have been implemented correctly, and do exactly what they are supposed to do. We will test the functions individually by running “unit tests” against them with different inputs and by inspecting the results. For example, your check_tower_blaster should return a bool value indicating if the given tower (or list) is “stable”, meaning the bricks are in perfect ascending order starting at the top (or left). We will test your function with a test tower (or list) as the argument and confirm that the function returns the correct bool value.
   As another example, add_brick_to_discard should add the given brick to the top (or beginning) of the given discard pile. We will call your function with a test brick and test discard pile as the arguments, then confirm that the top brick of the test discard pile is what it should be.

3. **Strategy – 5 points**
   Is your computer’s strategy something that makes sense? Please comment your code for that part of the homework very thoroughly, and explain your strategy for the computer.

4. **Usability – 3 points**
   This is going to be a subjective evaluation. You are making a game. A user who knows the rules of Tower Blaster should be able to play it without too much trouble. In particular, they should not have to read a single line of your code in order to operate the game.

5. **Style – 2 points**
   The usual stuff here. Style includes things like appropriate variable names, function names, clear comments, and general readability of your code. Every function must have a docstring defined. All non trivial lines of code must be commented. Style will always be part of your evaluation.
6. **Bonus – 10 points**
   Two player can play the game across 3 levels and a winner can be decided in the end.

Data Structures

In this HW you are only allowed to use lists and tuples. Do not use dictionaries, classes or any built-in python libraries (other than the random library for shuffling). Remember that the main pile, the discard pile, and the two towers (user and computer) are just lists.

Submission

Submit a single file called tower_blaster.py. Remember to put the following lines of code at the end of your file, as the entry point of your program. if __name__ == '__main__':

Your submission should include:

- Your name
- Your Strathmore University ID
- Statement of work. Ethics:
    - A list of resources you used and/or people you received help from (including TAs/Instructor)
    - A statement that you worked alone without help

Bonus Points/Advanced

1. Allow for two player to play
2. Modify the game such that you can have 3 levels