COMP10001 Foundations of Computing
Semester 1, 2016
Project 3 Description

— VERSION: 913, DATE: MAY 9, 2016 —

Changelog

- Updated to allow `player_data` as an optional argument to `score` (to support complete card counting)
- Updated to clarify that `prev_tricks` in the `play` function contains the tricks played in that phase, not the entire game
- Added the extra argument `phase_bids` to `play`, so the bids of the respective players are always accessible

1 Task Description

In this project, you will implement a program that plays a game called Oh Hell! ... or actually a variant thereof dreamed up by Tim, because he can't help himself. Your solution to each component of the project should take the form of a function, with function name and behaviour as defined below. All of your code should be written exclusively in Python3 and contained within a single file. The code should run successfully to completion within IDLE. The file should be named as follows:

```
program.py
```

We will play Oh Hell! with a standard deck of 52 cards. Each card has a suit (Spades, Clubs, Hearts, Diamonds) as well as a face value (numbers 2 to 10, as well as Jack, Queen, King and Ace). We will represent a card using a 2-letter string, with the face value followed by the suit. For example, the Queen of Hearts is `"QH"`. We will use `0` for the 10, and `A` for the Ace, so the 10 of Diamonds is `"0D"` and the Ace of Spades is `"AS"`.

A hand will be represented using a `tuple`, noting that the cards in the tuple are in no particular order. The following is an example hand containing five cards:

```
("AD", "AH", "JS", "2D", "7C")
```

Your task in this project is to write a program that can play Oh Hell! according to the rules in the Project 3 Game Description document. We will guide you through this process by building up the solution incrementally. You will be provided with a script to run yourself to help you assess the correctness of your solutions, and we will also make available a system to allow you to play against other students.

Q1: Write a function:

```
bid(hand, player_no, phase_no, deck_top, reshuffled = False, player_data=None,
    suppress_player_data=True)
```

that takes the following arguments:

(1) `hand`, a tuple of cards (each in the form of a string, as described above) that you have been dealt for the current phase, or in the case of Phase 1 or 19, representing the cards of the other three players

(2) `player_no`, an integer between 0 and 3 inclusive indicating player order for the current phase, where players play in increasing order of `player_no` value, and the player with a `player_no` of 0 has the lead for the first trick

(3) `phase_no`, an integer between 1 and 19 inclusive, indicating the phase number of the current hand

(4) `deck_top`, the top card of the deck, used to determine trumps for the current phase

(5) optional argument `reshuffled`, a Boolean indicating whether the deck was reshuffled as part of the deal for the current hand

(6) optional argument `player_data`, a user-defined data structure containing information about the current game state

(7) optional Boolean argument `suppress_player_data`, where `True` indicates that the function should return only the bid (and not `player_data`) and `False` indicates that `player_data` may be returned.

`player_data` is intended as a way of circumventing the need for any global variables in your code, and as such, the use of global variables will be penalised.[1] Note that there is no requirement that you make use of this argument. For the first call of `play` in a given game, `player_data` will take the form of `None`.

The function should return an integer between 0 and 10 inclusive, indicating the number of tricks you predict you will win in the current phase.[2] If you are wanting to pass `player_data` on to subsequent function calls, you should return a 2-tuple, where the first element is the bid, and the second element is the updated `player_data`. If you return an integer (and not a tuple), `player_data` will be set to `None` on the subsequent function call.

In the case of Phase 1 or 19, `hand` will take the form of a 3-element tuple of strings, representing the cards (in sequence of `player_no`) for the other three players. Note that you will not receive explicit information on your own hand in these phases.

Note that an implementation of `bid` which makes use of no strategy (e.g. returns the same value irrespective of the parameters, or randomly generates a bid) will be marked down for approach.

NOTE: *optional argument* means that it is optional to the *caller* of the function. As the *implementer* of the function, you must provide an implementation for **all** optional arguments that the specifications require.

Q2: Write a function:

```
is_valid_play(play, curr_trick, hand)
```

that takes the following arguments:

(1) `play`, a single card representing a potential play

(2) `curr_trick`, the tuple of cards that have been played in the current round (with the first card being the card that was led, and an empty tuple indicating that you have the lead)

(3) `hand`, the tuple of cards that you currently hold in your hand

Your function should return `True` if `play` is valid, and `False` otherwise.

NOTE: You can assume that all arguments to `is_valid_play` conform with the function definition (e.g. the first argument will also be a two-letter string which corresponds to one of the 52 cards).

Q3: Write a function:

```
score_phase(bids, tricks, deck_top, player_data=None, suppress_player_data=True)
```

that takes as arguments:

(1) `bids`, a 4-tuple, containing the bids of the players, in order of `player_no`

(2) `tricks`, a tuple of 4-tuples, each representing a single trick, in the order of play, starting with the lead card, and with `player_no` 0 leading the first trick

(3) `deck_top`, a string representing the card that was turned up on top of the deck at the start of play (which determines trumps).

(4) optional argument `player_data`, a user-defined data structure containing information about the current game state

(5) optional Boolean argument `suppress_player_data`, where `True` indicates that the function should return only the bid (and not `player_data`) and `False` indicates that `player_data` may be returned.

Note that this function will only be called with data from a completed phase, meaning that the phase number will be implicit in the number of tricks played.

Your function should return a 4-tuple of integers, representing the score for each of the four players (in order of `player_no`, as per `bids`). Note that the `player_no` of the leading player for a trick other than

[1] Global variables that are used as **constants** are, of course, strongly encouraged; but global variables that vary in value as the game progresses are strongly discouraged.

[2] Noting that in phase 2, e.g., a bid of 9 is not achievable (as there are only 2 tricks to be won in total), but that this will be accepted as a valid bid.

the first will be determined based on the preceding trick, and may be something other than than 0. If you are wanting to pass `player_data` on to subsequent function calls, you should return a 2-tuple instead, where the first element is the 4-tuple with the scores, and the second element is the updated `player_data`. If you return a 4-tuple (and not a 2-tuple), `player_data` will be set to `None` on the subsequent function call.

Q4: Write a function:

```
play(curr_trick, hand, prev_tricks, player_no, deck_top, phase_bids,
     player_data=None, suppress_player_data=True, is_valid=is_valid_play,
     score=score_phase)
```

that takes as arguments:

(1) `curr_trick` in the form of a tuple containing the cards played in the current incomplete trick (as per Q2)

(2) `hand`, the tuple of cards that you currently hold in your hand

(3) `prev_tricks`, which takes the form of a tuple of completed tricks for the current phase (each of which is a 4-tuple) as per Q2

(4) `player_no`, an integer between 0 and 3 inclusive indicating player order for the current phase, where players play in increasing order of `player_no` value, and the player with a `player_no` of 0 has the lead for the first trick

(5) `deck_top`, the top card of the deck, used to determine trumps for the current phase

(6) `phase_bids`, a 4-tuple containing the bids of the players for the current phase, in order of `player_no` (starting from `player_no = 0`)

(7) optional argument `player_data`, as per Q1

(8) optional argument `suppress_player_data`, a Boolean where `True` indicates that the function should return only the play (and not `player_data`), as per Q1

(9) optional argument `is_valid`, a function which implements Q2 and defaults to your solution to that question

(10) optional argument `score`, a function which implements Q3 and defaults to your solution to that question.

Your function should return a single card representing your next play, either as a string, or as the first element of a 2-tuple, with the second element being the updated `player_data`, to be passed as an argument to the subsequent call of `play`.

When we test your code, we will use our own implementation of `is_valid_play` and `score_game`, so that any errors you previously made will not carry forward in the marking.

A basic program does not need to made use of `player_data`. This is provided for more advanced programs to use in determining their strategy.

This question is substantially more open-ended than the first three questions. A basic solution that is able to play the game will be awarded full marks for correctness, but is not likely to be competitive and will be marked down for approach. Part of the mark for this project will be awarded on the basis of a competition between all student submissions, and a project that performs poorly in the competition will receive a low mark for the competition component.

As per the marking sheet, marks in this project are set aside for a competition. All working solutions will be played against each other in a tournament fashion, and marks will be allocated according to your game-playing program's final ranking, with the top-ranking student receiving full marks; all subsequent qualifying students will receive a mark > 0 proportional to their rank. Solutions that fail testcases will not participate in this competition, and will receive no marks for this component.

2 Bonus Question

The bonus question is based on a modified version of the game, whereby bidding takes place *sequentially* rather than in parallel, starting from the player who has the lead for the first trick. Also, in addition to bonus points for "making" a bid, there are variable penalty points for not making a bid (based on how far the bid was from the actual number of tricks won).

Write a function:

```
bonus_bid(hand, prev_bids, phase_no, deck_top, reshuffled=False,
    player_data=None, suppress_player_data=True)
```

that takes the following arguments:

(1) `hand`, a tuple of cards (each in the form of a string, as described above) that you have been dealt for the current phase

(2) `prev_bids`, a tuple of non-negative integers representing the bids of the preceding players

(3) `phase_no`, an integer between 1 and 19 inclusive, indicating the phase number of the current hand

(4) `deck_top`, the top card of the deck, used to determine trumps for the current phase

(5) optional argument `reshuffled`, a Boolean indicating whether the deck was reshuffled as part of the deal for the current hand

(6) optional argument `player_data`, as per Q1

(7) optional Boolean argument `suppress_player_data`, where `True` indicates that the function should return only the bid (and not `player_data`) and `False` indicates that `player_data` may be returned.

The function should return an integer between 0 and 10, indicating the number of tricks you predict you will win in the current phase. As for Q1, if you are wanting to pass `player_data` on to subsequent function calls, you should return a 2-tuple, where the first element is the bid, and the second element is the updated `player_data`. If you return an integer (and not a tuple), `player_data` will be set to `None` on the subsequent function call.

Note that the only difference in the function call over Q1 is the inclusion of `prev_bids` as an extra argument.

Based on how close your predicted score is to the actual score your player gets in that hand, you will receive bonus/penalty points as follows:

Margin between predicted and actual score	Penalty points
0	5
1	0
2–3	-5
4–5	-10
6–7	-15
8 or more	-20

These penalty points are added to your score at the end of each phase. For example, if you predict that you will get 7 tricks for a given hand and in fact get 4 tricks, the difference between the two is 3 tricks, meaning that you receive an additional -5 points, culminating in a score of $4 + (-5) = -1$ points.

3 Assessment

The project will be marked out of 10, and is worth 10% of your overall mark for the subject. Note that there is a hurdle requirement on your combined worksheet and project mark, of 20/40 (of which this project will contribute 10 marks).

We will not accept late submissions. If there are documentable medical or personal circumstances which have taken time away from your project work, you should contact Tim via email at the earliest possible opportunity

(this generally means well before the deadline). We will assess whether special consideration is warranted, and if granted, give permission for a reduced submission to the project. No requests for special consideration will be accepted after the submission deadline.

This project should be completed on an *individual* basis. While it is acceptable to discuss the project with others in general terms, excessive collaboration and any sharing of code are considered a breach of the University's Academic Misconduct policy (`http://academichonesty.unimelb.edu.au/policy.html`). Submission of your project constitutes a declaration of academic honesty and that your submission is strictly your own work (with the exception of the library made available to you). We will be vetting system submissions for originality and will invoke the policy for all involved students (whether you were the provider or recipient of the code) where inappropriate levels of collaboration or plagiarism are deemed to have taken place.

4 Playground

In order to allow you to test your player against other players, an online playground system will be made available closer to the submission date. This system will allow you to upload your program, which will then be played against 3 other randomly-selected submissions, and will give you a hand-by-hand transcript of the game. Exact details will be made available via the LMS.

5 Submission

Submission will be via Grok. Details of the submission process will be posted on the LMS.

6 Getting Help

Getting a bit lost in the details of the subject? Having problems with a particular concept or question? Have a burning question you want to ask more generally about Computing? The primary places to get personalised help with the subject are:

- The subject forums on the LMS
- Virtual drop-in help centre on Mon–Fri
- Weekly workshops
- After lectures
- Revision lectures, every second Thursday
- During Andrew and Tim's office hours

And as a general piece of advice: if you are feeling lost, get help now, rather than sitting on it and getting irrevocably lost as the subject continues to plough ahead.

7 Changes/Updates to the Project Specifications

We will use the LMS to advertise any (hopefully small-scale) changes or clarifications in the project specifications. Any addendums made to the project specifications via the LMS will supersede information contained in the hard-copy version of the project.

8 Important Dates

Release of Project 3 details	2 May, 2016
Deadline for submission of Project 3	25 May, 2016 (9:00am)