|epcc|


# Software Development
# Development and Reflection

B087928

March 31, 2016

# 1 Introduction

We shall report on the development of a deck building card game. We include a summary of the changes and improvements to the code and how these changes benefit the user or future developers of the code as well as the overall code quality and efficiency. We further reflect on the development process and how this development compared to our workplan and expectation of risks and unpredicted difficulties. Finally we comment on future enhancements that could be implemented to the code.

We used the online repository github for the development process. Once each new improvement had been implemented the functionality of the code was tested and the new version committed to the repository such that if at some point a new version of the code failed a test we could compare it to a previous working version and therefore losing at most only what we'd changed since last commit. The repository may be found at:
`https://github.com/bouracha/Software_development`.

We present improvements in order of implementation. We hope that this will give the reader an insight to the development process as well as a more understandable flow of information.

# 2 Code Improvements

## 2.1 Variable naming

It was considered important to do the small job of changing variable names before any modularity was introduced to the code. Most of the variable names are self explanatory but some are unclear. We shall discuss the main changes. The first that were considered unclear and therefore changed were pO and pC. These objects were renamed 'Player' and 'Computer' respectively.

Changing variable names makes the final code much more readable to future developers. 'act' and 'bv' were changed to '`action`' and '`buy_action`' which were more consistent with their purpose.

The final significant changes were 'pG' and 'cG' to `play_game` and `playing_game` respectively. These are boolean values that determine if the game should be played, continued or ended. It is unclear what the original variable names stand for.

## 2.2 Computer's move modularisation

We can consider this as one of the major stages of the game which is governed by a large block of independent code- any change to this code effects the way the computer moves and only the way the computer moves and is therefore the most perfect piece of code to move to another module called 'computer_move. This code is then simply called as a function from the main code body with as follows.

```
computer_move(card, central_line, Player, Computer, aggressive, random)
```

## 2.3 Functions module

We made another module named functions.py which contained small functions that were used throughout the program. These functions were:

### 2.3.1 `def draw_hand (Player, random)`

This code takes as argument 'Player' and draws that player's hand. This is used twice consecutively in the code with arguments 'Player' (for the user's hand) and then 'Computer' (for the computer's hand).

## 2.4 Player options modularisation and debug

A module called `player_options.py` was introduced. This module contains the functions that get called when one of the player options are selected. `def End_turn` is called from main when the user ends their turn and also from computer_move when the computer completes its turn.

This module contains four functions; one for each action (with the exception of attack since this code is only two lines) they are:
`Play_all(Player, card, money, attack),`

`Play_card(action, Player, card, money, attack)`,
`Buy_card(Player, central_line, card, money)` and
`End_turn(Player, card, random)`

As a result of this module we have a much more readable main code. All the actions may be seen at once and it is easy to make further investigation on each one by visiting the specific code.

During the modularisation of this code a bug was detected. Every time an individual card was played the values of money and attack were increased by the values on all the cards in the active area. This clearly makes the game very unfair and was fixed by removing the for loop that summed over all all active cards and adding a line that just added the value of the card most recently added to the active area.

### 2.5 Print statements and presentation

A new function was introduced to the `players_options` module called `print_values`. This is called just before the user is asked to enter her choice. Moving these print statements to this module makes the main method more readable. Furthermore, some extra spaces and framework was added to the print statement to make the interface more easy to interact with.

The `draw_hand` function was called from the from the `End_turn` function, this is the third call to this function.

Print statements that showed available cards and the supplement just after the computer moved were removed as they were believed to crowd the interface- the user may see these values buy selecting the Buy option.

## 3 Review of Process and Extra Improvements

So far we have presented, in order of implementation, improvements to the code. We shall now report on a few improvements that are largely extensions of previous ones and were more

easily located once the original plans had been executed. We also review how our process allowed us to come to notice these further improvements.

Once modularised, the code was a lot shorter and the playing game while loop was much more well defined. Creating Modules for code that had a specific function, for example: computer_move, was a very useful first step. Even though the collective length of the code was the same it was much easier to understand the work flow when the main was so much sorter. Therefore changes to the code and desired improvements could then be very easily determined and implemented safely.

It was clear now that the draw_hand function is called before this while loop which means that the hands must be drawn again at the end of each turn. Instead we could remove the extra draw_hand calls and move the ones in the loop to be just before the computer's move and player's move. This also removed a bug since during one of the inline coded computer's draw_hand it only reshuffled the player's discard pile. Using a function for this task made it very easy to see that there was no bug anymore.

The section of code that ended the game had code checked if player's or computer's health was less than 0. The game was also to end if there are no more central cards available. In this case the victor was determined from who had more health remaining. If this was not enough to distinguish a winner the victor is whoever has stronger cards remaining in their deck. However this final piece of code was not implemented correctly. The original code only declared two variables 'pHT and 'pCT', the meaning of which are unclear, and printed "Player One Wins on Card Strength" if pHT > pCT and "Computer Wins on Card Strength" if pCT > pHT. The following code was written to replace this section.

```
player_card_stregnth = 0
computer_card_stregnth = 0
while True:
    if (len(Player['deck']) == 0):
         random.shuffle(Player['discard'])
         Player['deck'] = Player['discard']
         Player['discard'] = []
    card = Player['deck'].pop()
    player_card_stregnth += card.get_attack()
    computer_card_stregnth += card.get_attack()
```

```
if (player_card_stregnth > Computer['health']):
      print "Player Wins on card strength!"
      break
if (computer_card_stregnth > Player['health']):
      print "Computer Wins on card strength!"
      break
```

Now in the case that the central line is empty and both the player and computer have the same remaining health then the victor is determined by determining who would win if the game were to continue: in this case the players would just be playing their cards and attacking so the above code calculates whether the user or computer's health would reach 0 first. Finally the entire block of code relating to turn ending was made into a function should_game_end and placed in the functions module.

Our original workplan was conservative. It was very difficult to determine the workflow from the original code so the most obvious improvements to be implemented was modularisation. It was predicted that further optimisations will become apparent after modularisation. This was found to be the case and hence justified having a conservative workplan that although wasn't kept to perfectly gave the work and good structure and most important gave us a chance to consider the order in whihc the work must be carried out. Our time was proportioned well and we could complete our initial plan ahead of schedule and work on further improvements. Since there were many improvements to implement categorising them as we did and dividing our time by taking into consideration the amount of code that relates to each optimisation and hence the amount of expected problems with implementations was a successful and safe way of implementing our improvements.

Frequent commits to revision control made the development process very safe.

# 4 Future Enhancements

Despite our many improvements there is still a lot of changes that can be made to this code to improve the quality. There may still be bugs in the code that were not found by our tests and there are also better ways of doing things that was not part of our focus.

I believe the main area for improvement is in the computer move. In our development we moved this section of code to a new module and also added function calls to draw_hand and End_turn. This is believed to be only part of the improvements that may be found in this section of the now modularised code. Since there are no functions in the main chunk of this section and many nested loops it is also the most likely location for more bugs. Thus further development would begin by attempting to break up this code block into function calls, we saw how this allowed us to detect bugs in the draw_hand function.