Software Development Plan and Risks

B087928

February 5, 2016

1 Introduction

When we develop software we aim to think further than a code that simply performs its function- although this is an extremely important aspect. We consider functional goals, that the code does what it's supposed to, and performance goals, that it does what it's supposed to well. In order to ensure we develop code well it is necessary to have a good development process.

In this coursework we have been presented with a python code which plays a Deck Building Card game. The code works and does what it is supposed to do and there are no obvious bugs. However the source code is very poorly written. Poorly written code is difficult to understand and further develop; both for the original programmer and anyone else working on the software. In this report we aim to address the most severe defects in this code design and suggest possible solutions. Furthermore we shall create a development plan to correct these defects and improve the code. We shall ensure that this development plan is carefully scheduled such that we shall have completed the desired software by the deadline of 25th March 2016.

2 Code Issues and Proposed Improvements

We shall now consider the largest issues with the code. Most of the problems relate to modularity in some way as there is no modularity implemented in the code as of yet.

Computer's Move The code presented is more than 400 lines long. Once we look through these lines the most noticeable design flaw is the lack of modularity. All the code is contained in the main function which makes it look cumbersome and difficult to read. This immediately suggests that our code would be more readable if we made separate modules for specific tasks, for example we may have a function for the computer's move and call upon it in main instead of having this extra large block of code. Lines 206 - 284.

Draw Hand It is also very noticeable that there are several blocks of code that perform almost the same function but for the name of a variable used. For example consider the following code snippet taken from line 56-62 of the program:

```
for x in range(0, pO['handsize']):
    if (len(pO['deck']) == 0):
        random.shuffle(p0['discard'])
        p0['deck'] = p0['discard']
        p0['discard'] = []
    card = p0['deck'].pop()
    p0['hand'].append(card)
```

This code draws cards from user's (p0's) deck to fill her hand. If the deck is empty it reshuffles the discard pile and this becomes the new deck. This code is repeated in line 64-70 except with pC (the computer's hand) instead of p0. We could reduce the size of main by writing a function to perform the above code that takes the player as an argument ie. p0 or pC.

Print Actions At several points throughout the code it is required to print out values such as the Health of the players, players' hand and active cards. It would be more helpful if we had a function that prints these details. This function should take, as an argument, p0 or pC. To implement all print statements in a single function module shall take a lot of time since there are many print statements throughout the code.

Player Options On the user's turn they will be prompted by the following request:

```
"Choose Action: (P = play all, [0-n] = play that card, B = Buy
    Card, A = Attack, E = end turn)"
```

The code body that governs what happens when each of these actions are selected should be modularised for simplicity. This will make the main function very clear and concise. The reason why this is so important is that once successfully implemented then it is easy to predict what we expect to find in each module ('play all' etc) from glancing at the main without being overwhelmed by the body of the code. If any details for what happens when any of these options are selected needs to be examined it will be easier to see further details from the modules. Lines 106-204.

End Turn The computer's end turn should perform the exact same lines of code as it does in the module to end the user's turn (option 'E' when the user is choosing an option). Thus when the computer ends its turn (line 286-299) it should simply call this function with argument pC rather than p0, note this argument will be the variable pC in the following code snippet.

```
    if (len(pC['hand']) >0 ):
       for x in range(0, len(pC['hand'])):
          pC['discard'].append(pC['hand'].pop())
    if (len(pC['active']) >0 ):
       for x in range(0, len(pC['active'])):
          pC['discard'].append(pC['active'].pop())
    for x in range(0, pC['handsize']):
             if len(pC['deck']) == 0:
                random.shuffle(pC['discard'])
                pC['deck'] = pC['discard']
                pC['discard'] = []
             card = pC['deck'].pop()
             pC['hand'].append(card)
```

Game Setup To set up and initialize the game players' healths and decks are reset to default as well as the central area and deck. This takes many lines of code and appears twice in a code body. It will be very sensible to modularise this and will cut down the code size massively.

Available Cards The central 'available' cards are shown before we have been asked to play. While it might be advantageous to see what cards would be available to purchase before committing to playing the game it is a bit confusing to a novice player. I believe it is best only to see this when we have money to potentially buy cards. Thus printing available cards after the computer's turn should be removed. Line 302-308. As well as making a less confusing interface, this will make the code shorter as we only need print active cards as part of the buy module/function.

3 Development Plan

Now that we have identified the major problems in the code and proposed how we'd like to fix them we must make a development plan. We shall attempt to estimate the time and effort required for each section of improvement.

The improvements we have suggested are simple and, in principle, won't take a huge amount of time. However, as these improvements are implemented we are likely to either uncover other problems that we shall face or further adjustments that are necessary to justify what we have already done. Thus the following time estimates take into consideration the likelyhood of further work - than what has been explicitly stated- being implemented. This shall be further illustrated in the risk assessment section.

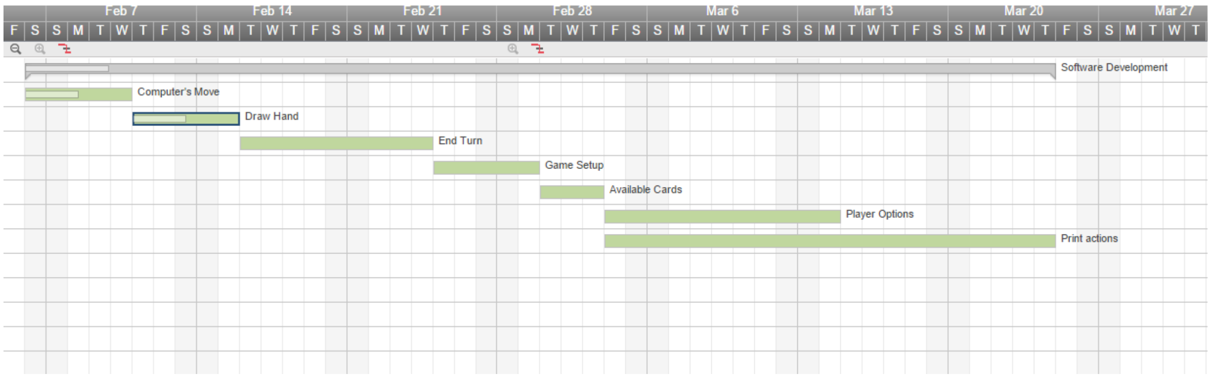

Figure 1: Gannt Chart. This shows the estimated and scheduled time that shall be spent on each of the development sections that have been outlined in the previous section.

4 Risk assessment

When we develop code there is always risk involved in the development process. We may have risk of schedule slippage or quality slippage. The Gannt Chart shall help keep us on schedule.

We shall consider a qualitative risk assessment by compounding the probability of a risk and the impact it will have in time to complete the project for each of the development sections we have outlined. We shall then compare if this justifies our gannt chart in the previous section and if we are likely to complete development on schedule.

We have summarised the risk for each section:

Computer's Move Risk here is very small. We don't need to make any change to the body of the code to put this section in a module. Furthermore, this module shall only be called once, so a problem should have a low impact and be easy to debug. Status: Probability: Unlikely; Impact: Negligible.

Draw Hand It should be easy to implement this function with only a small risk of making a mistake as the code to draw a new hand is already written. We need only change it into a function that takes argument pC and p0. Although this risk is low the impact would be slightly higher if a mistake were to be made- for example if both players were drawing from the same deck- as it will be difficult to detect and would make the game wrong. Status: Probability: Unlikely; Impact: Moderate.

End Turn We are making several new modules here so the probability of something going wrong is slightly higher with a moderate impact if something goes wrong. Status: Probability: Possible; Impact: Moderate.

Game Setup We shall make a module with a large block of code which should only be called at the beginning of the while loop that starts the game. Since this is such a large block of code it is likely that there are lots of small changes that shall be required to be made when we remove these two large bodies of code. Although this gives the risk a high probability, it should have a low impact on the amount of time required to fix all the small subsequent changes. Status: Probability: Certain; Impact: Negligible.

Available Cards This shall be mostly removing unnecessary print statements and likely other small presentation issues. Thus we are likely to do more than we suggest in the previous section but this can be adjusted depending on schedule. Therefore the risk of needing to do extra risk is high but very negligible. Status: Probability: Probable; Impact: Negligible.

Player Options Creating modules for each option should be low risk, however it is likely to give chance for further improvements within these modules and therefore we have a higher risk involved. Status: Probability: Probable; Impact: Moderate.

Print Actions There are print actions all throughout the code. Therefore it is very likely that we shall change something we don't want to while trying to modularise and simplify this aspect. This is especially high risk when we have print actions in the Player Options modules. This is why these two sections shall be worked on simultaneously. If a problem occurs it could have a significant impact on what the code does. Status: Probability: Probable; Impact: Severe.

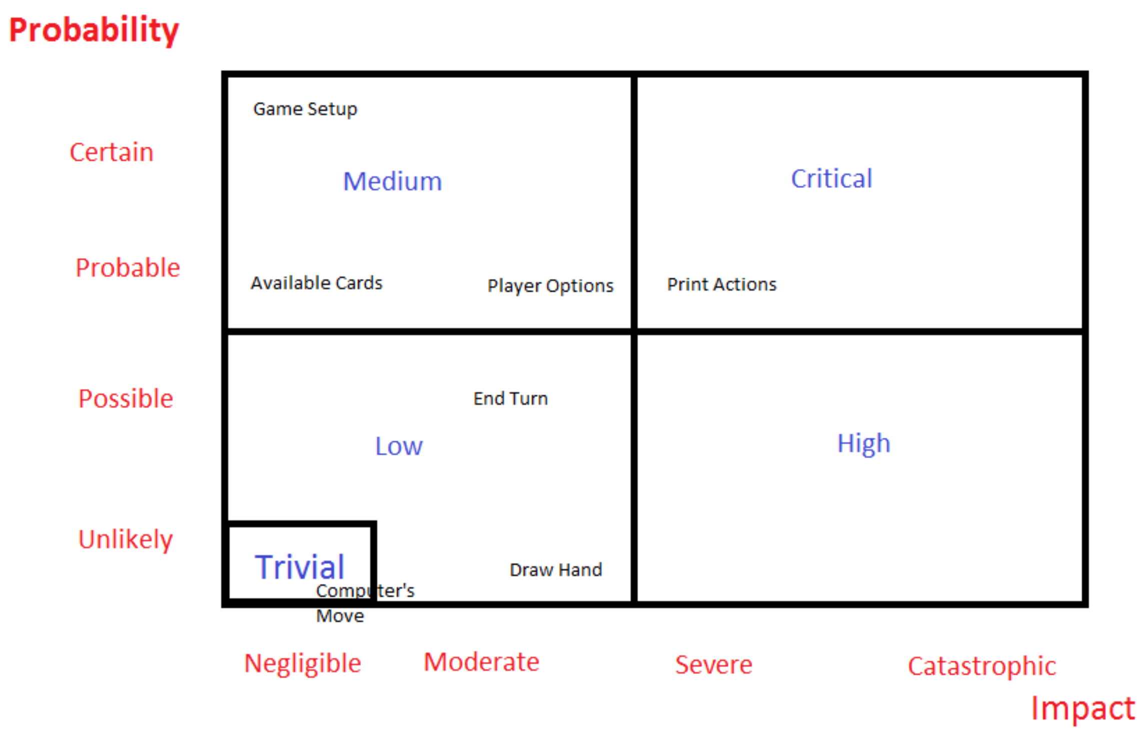

Figure 2: Risk Chart. We can see that we were correct to have allowed most time for the high risk/impact processes such as Print Actions and Player Options.

5 Conclusions

When we undergo a software development project it is vitally important to make a plan. We need to have a clear idea of the functional and performance goals of the software. Otherwise, even if we are excellent programmers, we have no idea when we expect to be finished or what our final software shall do. Furthermore, without a plan we are not prepared for what might go wrong and slow down or potentially break the development process.

We produce a Gannt Chart so that we may have a good idea of when we must finish each section of the development process and which tasks are to be completed simultaneously. This shall help keep us on schedule in our design process. To produce Gannt Charts it is important to take into account the risks involved in each section of the development process.

With a clear outline of our goals, good scheduling and an accurate assessment of the risk involved it is possible to ensure that we complete software development in the time given.