|epcc|

Software Development Assignment 3

B083194

March 23, 2016

Contents

1 Introduction

In this assignment, the changes from assignment 1 have been implemented, and the schedule from these implementations has been compared to the schedule proposed in assignment 1. The changes made have line references, and they are referring to their location in the original code. When changes were finalised they were uploaded to Github so that progress could be monitored, and compared to the aforementioned schedule. The link to this repository is `https://github.com/karldelargy/SDAss3`. Some unanticipated risks and areas for further development are discussed at the end.

2 Summary of Changes

This section will go through the changes made to the code, along with their associated benefits. Some of these benefits improve the code, making it more maintainable and reusable. Others are aimed at improving the overall game experience. Which bracket a change comes under will be highlighted as appropriate.

2.1 Defective Adding System (lines 124-129)

Change(s): There are two options for playing cards: Play all or play each card individually by selecting the corresponding card index. If the second option is chosen for playing more than one card, the attributes are added cumulatively each time. This greatly overestimates the attributes of each player. This problem was fixed, making the game add up correctly.

Benefits: A game with improper functionality may be off putting to potential players. This fault made the game too easy, as you have an incredible advantage over the computer. Removing this fault brings it back to a level playing field, increasing the competitiveness of the game. This change improves the overall game experience.

2.2 Hand Strength Deciding Victor (lines 324-332)

Change(s): The code attempts to compare hand strength between the two players, but before the comparison, the hand strengths of both the computer and the player are set to zero. This makes the comparison useless. The code has been change to loop over the players cards (all of them) to determine the hand strength instead of presuming a draw each time. It should be noted that the game was not clear on what hand strength was, so it was assumed here to be the sum of cards attack and money points.

Benefits: The player who has acquired more attribute points will now win the game, meaning it is more likely someone is victorious. This change improved the overall game

experience. The game details didn't mention this feature of the game, so it could be that this code was just dead code. Another approach may have been to remove this segment of the code.

2.3 Self Documenting Variables (throughout)

Change(s): Many of the variable names were acronyms, or made entirely no sense. These names were changed to be self documenting variables. These included:

1. pO => playerOne
2. pC => playerComputer
3. pG => playGame
4. cG => createGame
5. central => centralLine

Comments were also added to the code.

Benefits: Self documenting names make the code much more understandable, and therefore reusable and maintainable. Comments help us understand the code, which is particularly helpful when we haven't looked at it in a while, or new people are looking at it. This change has improved the code.

2.4 Removed Dead Code (lines 4 and 8)

Change(s): The code mentions `clan` on two occasions when it is not used again for the rest of the game. These pieces of code are called dead code, and are removed from the program.

Benefits: Removing dead code cleans up the program, improving readability. Dead code may also cause confusion as programmers may have difficulty understanding why it is there, but may be cautious of deleting code in case of unforeseen consequences. This change has made improvements to the code.

2.5 Unwanted Continuation (lines 82-84 and 335-336)

Change(s): If a player is asked if they want to play a game, and they decide they do not, the computer still asks what type of opponent they would like to play. This question is redundant as they have expressed their desire not to play the game. This has been changed to exit the game when the player does not want to play, rather than continuing to the next part. The game does not inform the users of what keyboard input will select yes or no, so this has been made clear.

Benefits: This could frustrate players who just want a quick exit. One of the features of a good game is clear exits [1], and players may be mislead into thinking that the game ignored their request to not play the game, making the game seem as though it has bugs. All these symptoms are fixed through fixing this mistake. Users may get stuck when trying to figure out Y means yes and N means no, especially those with little programming eperience. This issue has been resolved. These changes have made improvements to the game.

2.6 Displaying Remaining Money (lines150-152)

Change(s): When purchasing cards the remaining money is not printed to the terminal. This was changed to print each time a card is bought.

Benefits: This can be extremely frustrating as players depend on what they can afford, and it can be difficult to budget as remaining money gets wiped after your turn. Printing the remaining money will mean users do not have to exit the buying window to check their balance, and move back equipped with that information. In other words in helps reduce cogitative overload and unnecessary moving backwards and forwards between windows. This change has made improvements to the game.

2.7 Slowing Down The Output (throughout)

Change(s): Throughout the game output is printed to the screen. This output is printed out in one go for each stage, and in some cases, such as the computers turn, this takes up a lot of space. `sleep()` functions were added to slow down the output.

Benefits: This makes the output more digestible for humans to process. When the computers turn was dumped out in one huge chunk, it was especially difficult to pay attention to it. The delays are aimed to achieve timings close to how the cards might be dealt in real life. This change has made improvements to the game.

2.8 Exit Options (lines 101 and 150)

Change(s): There are no options at any point in the game for exiting. These were added in.

Benefits: The game should provide ways to exit the game for players who have finished playing before the game is over. These are a very important usability features [1]. Players can get frustrated if these are not frequent, never mind non-existent as in our code. This change has made improvements to the game.

2.9 Spelling Mistake (several instances)

Change(s): Acquisitive is spelled "Acquisative", this has been corrected.

Benefits: Although this is minor in the grand scheme of things, it still presents itself as mistake which may represent an unfinished quality to the game. This change has made improvements to the game.

2.10 Modularisation(throughout)

Change(s): The code was contained in one big file. This is poor programming practice. The code has been changed to be made up of smaller independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality [2].

Benefits: The separate modules can be updated independently. It improves the readability and maintainability of the code, and follows good programming practice. This change has made improvements to the code.

3 Reflection and Changes

The time schedule from the first piece of coursework is demonstrated by the Gantt Chart from assignment 1. This is displayed in Figure 1.

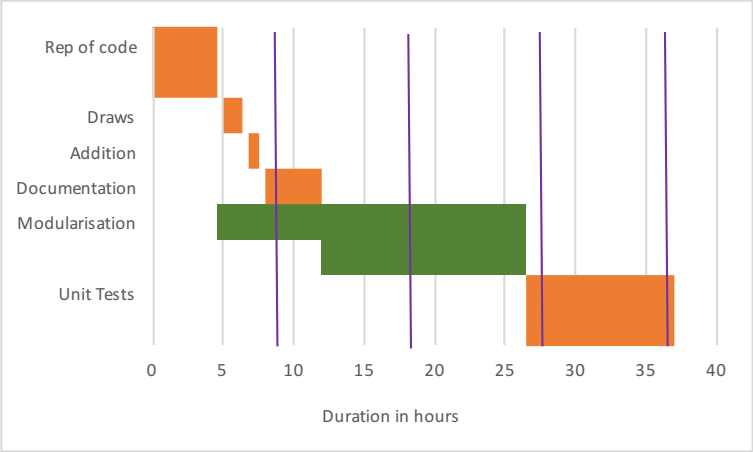

Figure 1: Gantt chart of estimated schedule from assignment 1. Purple lines represent a new day.

The first thing to note is that unit testing has been left out. Although there was thorough testing in the form of playing the game to check all the features, I left myself without enough time for proper methodical unit tests. This can be put down to poor planning and time management.

Ignoring unit testing, the plan suggested the changes would take around 3 days to complete. This included working simultaneously on modularisation and other tasks concurrently. It didn't quite work out like that. Most of the tasks were performed together on the first day whilst everything was in the same file. This made it easier for tasks like changing variable names where it was helpful to view the code as a whole. Once these tasks were completed, it was time to begin modularisation. This process again took approximately a day, around half the estimated time. The project was "completed" with a day to spare. However, Parkinson's Law states "Work expands to fill available time" [3]. This proved to be true. The next day was spent tinkering with the code, testing every possible scenario and making minor changes.

Some changes could not be made, such as removing some of the code when the game is played again. Upon initial inspection, a lot of it seemed unnecessary, however the code would need a complete rewrite to cut this out. Although, other tasks were added in, such as adding options which allow users to exit the game. One thing that becomes clear is that the plan enviably changed as it went along, and the schedule has to be adapted. Estimates need to be continually estimated until the project is over. Prior experience is extremely important in estimation [3], and considering I have no experience, the estimate was not terrible.

Planning is a powerful tool which has the potential to save projects. They help the relevant people decide if a project is financially viable, and provides customers an idea of the product they will receive. It gives the people working on it an idea of what stage they should be at, and milestones give a sense of achievement and progress. Estimates should be updated frequently, as it is difficult to predict everything that may happen. Incorrectly estimating can be as dangerous as no estimation, causing the project to go so over budget it is no longer feasible, and projects get discarded at a huge loss. In conclusion, all projects should have a plan, but they are only worthwhile if estimates are realistic.

4 Unanticipated Risks

There were several risks which were unanticipated in the plan, and were only discovered when implementing the changes from said plan. These are outlined below:

Risk 1: It is nearly impossible to reach a scenario where the main deck and central line run out of cards and health is equal at this point. This unlikely run of events was required to test whether player hand strength would be added up correctly to decide which player was victorious.

Risk 2: Some of the variable names which needed changing were difficult to understand, and may have been changed to a name which does not accurately represent that variable.

Risk 3: It is difficult to find all instance of dead code in a program, there may be more hidden within the code.

Risk 4: The code would need significant changes so that it would only accept either A or Q when choosing opponent, in hindsight, this risk was too great for the limited improvement it may make.

Risk 5: When modularising the code, it got confusing when trying to remember which modules and functions need to be imported. Figuring out which variables needed to be returned also caused difficulty.

Risk 6: The code at the bottom initialises the game if the player wants to play again. This code is largely a repetition of the initialisations at the start. The code is written in such a way that it would need a rewrite in order to remove this.

5 Future Enhancements

This project was about following the plan from assignment 1, and comparing the plan with how things actually played out. But as these changes were made, many further modifications which added value to the program became apparent. These are listed below.

1. **Unit testing:** Although this was actually part of the initial plan, unit tests would greatly improve this program as a software product. Unit testing is an important part of software development, and should be carried out as the project goes along. Software companies which produce buggy products will find themselves with a bad reputation, and hence sales will go down.
2. **UI improvements:** The user interface is fairly basic, and there are various improvements which can be made. Firstly the screen should clear when moving between windows, eg from buying window to the main window. Text based pictures could be used to make the game more interesting. The output in the terminal could be partitioned to make it more clear.
3. **OO:** The code could be improved to make it object orientated, with classes etc. This makes the code easier to work with, extendable and more flexible.
4. **Instructions and settings:** The game could include instructions which can be accessed through the main window, or before the game starts. The game could have additional settings, so regular users can navigate through the game quicker, such as turning off the sleep functions.

References

[1] University of Edinburgh, EPCC, Software Development, Lecture notes, Usability

[2] Modular programming, Wikipedia, https://en.wikipedia.org/wiki/Modular_programming

[3] University of Edinburgh, EPCC, Software Development, Lecture notes, Project Estimation