**Comp Science NEA**

Contents

[**Analysis** 1](#_Toc496253662)

[Introduction to Organisation and Client 1](#_Toc496253663)

[Outline of the Problem 1](#_Toc496253664)

[Investigation 2](#_Toc496253665)

[Analysis of Investigation 4](#_Toc496253666)

[Objectives 6](#_Toc496253667)

[Data Flow Diagram of Game 9](#_Toc496253668)

[Data Flow Diagram of Current System 10](#_Toc496253669)

[Research 11](#_Toc496253670)

[Potential Solutions 13](#_Toc496253671)

[Proposed Solution 16](#_Toc496253672)

[Proposed Solution DFD 18](#_Toc496253673)

[Documented Design 21](#_Toc496253674)

[Software 21](#_Toc496253675)

[Testing 21](#_Toc496253676)

[Evaluation 21](#_Toc496253677)

## **Analysis**

### Introduction to Organisation and Client

Mr X is a freelance graphics designer who wants to begin developing games. His background in graphic design mean he specialises in developing smooth and aesthetically pleasing interfaces. However, whilst he has been learning how to program and develop games on a fundamental level, his technical ability is still growing, and his mathematical ability is not as proficient as he would like it to be for some parts of this development.

Knowing me from secondary school, from being in the same IT class, he remembers our contrasting interests – his in the more design focused aspects, mine in the more technical side – and has contacted me to see if we can collaborate on this project of his.

### Outline of the Problem

His first game is going to be a casual (online?) Standard deck card games game. The most technical aspect of this project is the development of a competitive AI, which the user can effectively practice against, with varying levels of difficulty. The developer is looking to delegate this objective to someone who has a deep understanding to the mathematical aspects and concepts underlying. Specifically, the developer wants the AI to have different general personalities (in a game context: aggressive, defensive), as well at higher difficulties to have varying personalities based on the situation.

For example, one of the standard deck card games which will be a part of the game will be poker – in this context, the client would like the AI to have a personality where it is aggressive - raises a lot to attempt to intimidate the opponent for example - and another where it plays more passively – folds more often, and raises less often. In addition, when at the higher levels, the AI would play more aggressively when it has a higher bank, or it thinks the opponent is on tilt, and in other scenarios it would play more passively.

The current system, if it were to be developed by the client, would consist of simple logic, and transparent decisions, where the AI would eventually become predictable and the interest from the end users in the game would drop very fast. Under the current system, the client would have to spend a lot of time explicitly programming the AI, in order to accommodate for specific scenarios, to emulate the nuanced decision making of a human. As a result, the client would like the AI to be competitive and a challenge for the end user, as well as this the game development deadline is 6 months, consequently, the client requires the development time to be kept reasonable to this deadline.

### Investigation

**Interview with Primary Client – Mr X**

**Specifically, what is the game you are thinking of developing and which demographic are you aiming it towards?**

It’s good to talk to you again after all this time! I hope you are doing well.

Well, I have an interest in games like poker, I just find them so fun! However, a side me of dislikes the potential gambling aspect in the sense that someone could destroy themselves over a game, and sometimes people just have not had enough practice to start playing for real – my goal is to develop a game for these sort of people, more of a casual base. I just want people to have fun, and I hope to keep them safe whilst they do it, there’s a bit of a vocational aspect for me personally.

**What are the central focuses of the game you are trying to develop?**

You know graphical design is where my strengths lie, so, I want to practice where the skills overlap, so I am thinking more of a focus on the UI. But, I also realise how technical this field can get, so I want to start learning the basics of the logical process, and some parts of web development. The catch is, with a game like this I do not want some of the more casual users to be put off by pressure of having to play against other real players, if they do not want it, so, I want to develop an AI which they can play against, but I am currently struggling with this more than anything else in the project.

**From your statement, it sounds like you have attempted to implement this AI, which suggests that you have design elements in mind. What, at the moment, are the central design elements or features you would like for this AI?**

I think I have a pretty good vision of the AI. I want this AI to have dynamic difficulty levels – I want it to be competitive, but not exceedingly difficult, to the casual user especially, but I also want it to provide a challenge for the more experienced users. After all, in my humble opinion, there is no fun if there is no challenge.

Also, an exciting part for me is designing this game with a compelling caricature aesthetic, with characters who have striking personalities. As a result, I would like the AI system of this game to have differing personalities to reflect these characters. For example, I am designing a half-man, half-bull character to be in the game, the AI for this character should have a more aggressive personality than usual, to reflect these characteristics.

**What is the current system you have, or are in the process of implementing?**

Ahh, it’s not very good, but so far, I have a simple pattern matching system, where the AI checks the state of the game, and then it checks against conditions I have programmed in, it then follows the corresponding action. Just a long if-elif-else chain, really.

**What are the benefits of the current system?**

Well, although it is not as good as I would like it to be, it does work on some level, I suppose. It can play a simple game, and sometimes it is kind of fun to win, however, it can get really boring, really fast. The only real benefit from a developing standpoint is that I can easily add more conditions for it to check against.

**What are the drawbacks of the current system?**

The big problem, really, is that it is very easy to pick up on the patterns the AI follows, I’m not sure if I am a bit biased, because I know the conditions it looks for, as I am the one who made them, regardless, it is too predictable at the moment. I’ll give you a quick example, say you raise a lot of money, the AI will always fold, as it will think that you have a good hand because you raised so much. It is so easy to exploit! Just raise a huge amount every round, and you will always win! Also, the AI will only ever be as good at poker as I am, because I cannot program to check scenarios I am not aware of.

Lastly, although a secondary issue, the AI currently does not exhibit the personality types which I desire. Moreover, to program this in would require a lot of time, because, I would have to program a different reaction to each scenario – so, say I want to have 3 personality types, for each pattern the program recognises I would have to program a different decision or reaction for each of these types. Furthermore, the AI would require a lot of maintenance, due to the nature of having to account for a lot of different scenarios – even if one common scenario is not accounted for, the AI is exploitable and would require constant patching, lowering the overall quality of my product. It’s a lot of hassle at the moment.

**Are there any final points you would like to add?**

The development of this AI is stunting the progress of the development of the final product a lot more than I would like. Preferably I would love to collaborate with someone, so that I can focus on developing the skills I want to focus on. In addition, ideally I would like the AI to not have the current maintenance aspect which it currently has, the best case scenario would be for it to be self-sufficient – by learning from its mistakes, for example – so that it is of an acceptable quality, even on the launch date.

### Analysis of Investigation

From this interview it appears that the client would primarily like the AI component of the development to be delegated. Firstly, a quality of the AI which the client currently likes is the aspect of its extendibility – “I can easily add more conditions for it to check against.” whatever the new system will entail should keep this aspect of extendibility.

On the flipside, the main issues with the current system appears to be its effectiveness, specifically, the AI is “it is too predictable at the moment” and “It is so easy to exploit!”, from this, we can reason that the new system needs to be more nuanced so that it cannot be predicted quite as easily. The example which is given is that the “the AI will always fold, as it will think that you have a good hand because you raised so much”, so one specific improvement could be to add a calculation element where the AI puts a probability of the opponent bluffing, then proceeds to fold if they think that the opponent really does have a big hand – either from previous experience of the opposing players tendencies or from absolute probabilities. Specifically, this issue of nuance is exhibited in the fact that the current system will always make the same decision in the same explicit scenario.

The current system is programmed using a “a simple pattern matching system” where the developer has preprogramed the AI to recognise certain states of the board – supposedly the data is taken straight from the game, since the product is of his own development – and then execute the preprogramed action in reaction to this pattern.

The current system works like this:

State\_of\_hand <- get\_hand()

…

Get state of all aspects of game

…

IF Opponent Raises High THEN

FOLD

ENDIF

…

Check more edge cases/patterns

…

Bet <- calculate\_bet(current\_hand\_score)

The big aspect here is the checking of the different edge cases/patterns – all the nuance in the program relies on having a robust case checking system full of a lot of different patterns. As the user states, this system is difficult, because it requires explicit maintenance of each different scenario the game could be in, and if one case is not accounted for and it occurs often enough, then the entire game is exploitable, which ruins the quality of the overall product. From this, we can infer that a better system would to develop an AI which does not need explicit programming to recognise different features, so that it can implicitly adapt to each new scenario, and develop its own nuance.

On the flipside it is important to make sure that the AI has differing difficulty levels, so that the difficulty of the game matches the ability of the current user, so that the user and the bot can remain in a competitive game, so that the users of any skill can enjoy the final product. A subtle aspect of this will be that it is more important for the final AI to be able to play less skilled users than users of high skill, because the client states that he wants to orient the product to “more of a casual base” – in addition, users of higher skill would most likely be playing on professional online gambling websites.

Another big aspect of the AI which the user would like implemented, is the feature of “personality”. The client states that his game has a style of “compelling caricature aesthetic”. The example he gives is of a half-man, half-bull character. The consequence of this for the AI is that the AI should reflect these personalities in game – for example, the half-man, half-bull character would be more aggressive than some of the other characters.

### Objectives

1. Firstly, the new system should provide a fully autonomous AI which can play against one or multiple opponents in a game of Poker, with a response time of less than 30 seconds, and a competitive difficulty, for a casual player. To elucidate the somewhat abstract definitions of the behaviour of the AI, autonomous means:
2. The AI should be able to adapt to new situations, without having to be explicitly programmed to handle that situation.

Furthermore, competitive means:

1. The AI should not exhibit an overly predictable playstyle, to the point where it can be countered easily. For example, if it always has the same response to the same stimulus, it is too predictable.
2. If the AI is losing multiple times to a specific opponent, it should identify the style/personality of the opponent and adapt to this player, in order to provide a competitive game to this specific player. For example, this is more of an extension on point a – if the player keeps using the same strategy to beat the AI, the AI should not keep making the same failing move, but transition to more effective move.
3. If the AI is dominating its opponent too hard, it should automatically identify this and drop its difficulty. This means that the AI should remain competitive to the opponent, but should not dominate them, to the point where it is not fun for the end user to play the game; equally, if the user begins to dominate the AI, the AI should increase its difficulty to adapt to this player. Concretely, if the AI wins 10 or more games in a row, or less if the games were not competitive (ie the user lost is every round they played), then the AI should change its behaviour to play less effectively. Moreover, the user should also have the option to disable this functionality, as they may purposely want to play against its highest difficulty, because they find it fun.
4. The module for the AI should provide a very restrictive interface, in order to prevent the end user from destructively adding games to the training, or changing the behaviour of the AI in any sort of way.
5. The program’s interface should only be able allow the client’s program to send the state of the current game, and the only thing it should return is the AI’s next move.
6. In addition to this, it should provide a separate interface after each game, which allows the client’s program to send the record of the game, in order for the AI to be able to analyse it and add it to the AI’s training (discussed in a later objective). This interface should only be accessible after each game, and would need a security verification, to prevent any other scenario from adding a game to the training of the AI – this ensures “False games” (games which did not happen) do not influence the AI’s behaviour, so it only adapts to empirical experience.
7. An addition an interface should be provided which allows the user to configure the personality of the AI, externally from each game. The extent of this interface should include a preset list of options the user can select from. Moreover, this aspect of the AI should not be adjustable within the game, so this interface should be disabled whilst the AI is in a game.
8. Externally to playing the game, the AI needs to be configured to play with differing playing personalities:
9. The pre-set personalities which the user has requested are: basic aggressive, and basic passive.
10. The mechanics of the personalities should vary based on the difficulty level.
11. At the lower difficulties, these personalities should reflect emotions – for example, after losing a big hand, the program should emulate a level of “tilt” where the program would not perform as much as an optimal play, contrasting to if the AI had won a big hand, after which it would “be more confident” and be more likely execute the optimal play.
12. In addition to this, at higher difficulties, the AI should not have a preset personality, but a dynamic style which changes based on the state of the game; also, the program would be less inclined to exhibit an “emotional” play style – for example, it would be less inclined to go on “tilt” after losing a big hand.
13. There should be an interface to configure the personality of the AI outside of the game, however, this should be restrictive to only external to any game.
14. There is potential for the AI to be trained, if this is applicable, then the program needs to fulfil the following requirements in this regard:
15. The program should be provided to the user pre-trained, and ready to be implemented into the game that they are creating.
16. The AI should use each new game it plays to add to its training, meaning that each game it plays will add to the proficiency of the bot. However, it should only train itself after a batch of 50 games, for example, as to not be influenced too much by anomalous games.
17. An interface should be provided, to allow for the client to add their own store of training games, if they wish to train the AI further.
18. The training data, as well as its algorithm should not be stored with the AI, in order to prevent potential malicious users from attempting to change its behaviour.
19. If the AI plays enough games against a significant user, which dominates the AI, the AI should attempt to identify the playstyle of this player, and retrain itself to play against this style in particular, then apply this new training either to this player, or players with a similar style, if the training it already has is not adequate.
20. To fulfil the other requirements of adapting to the user, the program needs to have a method/interface for identifying each user. This can either be built into the program, or be provided by the client. This involves, at some point, the user being able to log in and be provided with a unique account, in order to identify them. This includes:
21. A sign up system, where the user can provide a username and password, to create an account.
22. A log in system, so that previously existing users can sign into the program, with a record of their previous games.
23. The username should be unique to each user, and this should be checked and ensured during the sign up process.
24. The password will have minimum requirements of at least one capital letter and a number, to ensure the user does not allow themselves to have their account compromised.
25. Both of these inputs should be sanitised, to prevent attacks against the system in place.
26. Relating to the last objective, keeping a record of past games is a good way for the end user to reflect on previous decisions and improve, however, there is potential for the AI to use this history of games to improve its own performance. As a result of this double need, there should be a method of storing data about each game played.
27. A record of each game should be stored in a database – with each record being an account of each game played.
28. There should be a relationship between three tables – a table of users, a table of a game; which will contain information about each move in the game, and lastly a table of games, connecting multiple users to a single game.
29. This database should only be writable by a central server which is hosting the game, every user and the AI should only be able to read from it. Additionally, the users should automatically have access to each game they are involved in, and then any other games which they download externally. This access should be locked whilst a game is in play.
30. As the client potentially would like the final product to be web-based, multiple instances and connections to this database and AI is possible, consequently, some safety features should be implemented to prevent possible conflicts. An example, of this may be the use of timestamp ordering or serialisation.

### Data Flow Diagram of Game

Regardless of the nature of the final system, the nature of the game with stay consistent. It is important to lay out the data flow for the game.



### Data Flow Diagram of Current System

**Level 0 DFD:**



As far as the current system goes, this as detailed as it will be. Within the process of “Calculate System Position Value and Generate Move”, a series of if statements are utilised to generate the calculation and move.

### Research

Poker is a game of incomplete information – the complete state of the game is unknown at any given time – unlike a game such as Chess or Checkers. Consequently, this makes designing an AI with concrete, non-flexible calculations and pure mathematical models a less optimal solution. This holds especially since the opponent for the AI will be a human player who will be able to spot these patterns and adapt their own playstyle, and exploit the AI, if it remains linear – this problem is exaggerated especially in the current system.

A decent method would be to find a database of poker hands, such as the UCI Poker Hand Data Set[[1]](#footnote-1)(dataset of hands for 5 card draw, over 1 million instances) or Michael Maurer's IRC Poker Database (University of Alberta), then use statistical analysis to determine the common patterns or behaviours, in general. The biggest issue with this is that the data I have found so far is not contextualised, and the dataset does not contain the moves made by each player. For example, the data could have come from a high rolling club in Las Vegas, or a million instances of different kitchen table games, as a result, it is hard to add any context to the analysis of the data. In addition to this, the most the data could be used for is to analyse which hands are most likely to win – it does not provide any information for general player behaviours; this issue which stems from this is that using solely data analysis to calculate the behaviour of the system is that it may become too linear and play in one playstyle – for example, only betting when playing with a big hand. In addition, these datasets will generate a weakness in the system in the sense that it will not be able to reason about other player's behaviours, and adapt to them, whereas, some human players will be able to reason about the system's behaviour of using purely statistical information to inform its play, and perhaps outplay the system too easily.

Regardless, the fact that there are datasets already available makes it easier to provide some sort of baseline to the system I will develop, and still may be useful. Some concepts to apply to this information may be Bayes Theorem, and Nash equilibrium; these are game theory ideas which could be applied to my system to make it more effective at playing the game. Beyond that, within the game, decision trees and minimax trees to use within the game, despite incomplete information, these can be used to model possible future scenarios and decide which move may lead to the highest value position.

Another aspect which would be explored is the possibility of using random states to explore the trends of Poker, as an incomplete information set from any one game makes it hard to provide concrete and reliable analysis. One such method may involve using the Monte Carlo Simulation[[2]](#footnote-2), and generating my own dataset via an AI simulating many games against itself. Whilst this does, to a degree, amend some aspects of the system not being able to analyse aspects of opponents behaviour as the AI could be preprogramed to play in a certain way (such as tight passive, etc. Discussed later), and then the actions taken by each playstyle can be analysed, and extrapolated to assume that a human opponent would behave in a similar way, if they were to follow a similar playstyle; the problem is that a human opponent may not play exactly the same way, or any of the ways that the pre-programmed AI may play, in addition to this, this adds an extra dimension of initial opponent behaviour analysis, in order to determine which style category they may fit into.

In addition to this, another possibility could be explored fitting with the idea of exploring random initial conditions. Neural networks have been used in conjunction with an evolutionary algorithm, or reinforcement algorithm and made to play against itself to develop a playstyle which emulates a strategy generated from a metagame of playing against other AI, rather than human opponents[[3]](#footnote-3). Utilising neural networks may be a useful way to find an optimal behaviour to this problem, with the case of incomplete information, without the need for unreliable calculations based on speculation on the part of the system (as the system in general would not be effective at analysing behaviours and extrapolating this to value of future hands). This could be implemented in different ways – for example, a baseline weight for the system could be developed, and then used within the game, but then developed further against new opponents, in order to generate more effective weightings for their particular playstyle.

There are a few different options to implement a system like this. One possibility is to use a matrix library, such as NumPy[[4]](#footnote-4), and then design and implement my own neural network. This is a viable solution, as long as the matrix library in question is efficient in its operations. This would allow me more freedom in my implementation of the network and applied algorithms. Another option would be to use an external library which has prebuilt neural networks, and then adapt this to my problem, an example of a library which is capable to use for this is PyBrain[[5]](#footnote-5). Both of the mentioned libraries are for Python.

According to Pokerology.com[[6]](#footnote-6), in terms of "personalities" or playstyles, there are generally two spectrum which a player may fit on: Tight vs Loose, and Passive vs Aggressive. Tight players are defined as players who, in general, only play when they have a strong starting hand, and loose players, in contrast, will play more with weaker hands. In addition to this, Passive players tend to fold more often, and call/check rather than raise when they have a strong hand; following this, aggressive players tend to raise more often and risk more chips.

These can then be combined into more specific general playstyles: for example, a Tight aggressive player (colloquially labelled as a "Shark") tend to not play for many pots, but when they do they try to maximise their opportunities when they have a stronger hand. Moreover, as humans tend not to be linear entities, their playstyle may change in accordance to their emotion (for example, a player who plays worse after losing many pots in a row may be said to be on "Tilt"), or they may purposely change their playstyle to fit the table they are playing at – consequently, there is an extra dimension of Tricky vs Straightforward, where a straightforward player is more likely to fit better into the theoretical description of their general playstyle, whereas a Tricky player may change it up a lot more. In general, according to this source, aggressive playstyles tend to generate more revenue than passive playstyles, where Tight Passive players is tagged as the worst playstyle, putting their playstyle down to "being scared". Whether this may or may not be the case, all these playstyles are relevant to this problem, because the final product may be suitable for a user who is new to the game and may be inclined to play a more "Tight Passive" style, as they are still learning the game, as a result, this needs to be taken into consideration. Moreover, these concrete examples of different playstyles could be mapped to different personalities, which is one of the objective of the client.

### Potential Solutions

**Extension of the Current System – Pseudo-Manual System**

One potential solution would be to extend the current system. One of the main issues with the current system is that the system depends on the knowledge and understanding of the game of the developer. Consequently, the current system could be improved with deeper patterns to check and a more nuanced patterns to check for.

For example, the current system has the drawback of being too predictable, or its behaviour is too linear, to amend this deeper calculations could be used. Using the example the client gave me, the system, at the moment, will always fold when presented with a large raise, as it determines that the amount that the user bets is proportional to the value of their hand – ie it does not take into account the possibility of bluffing. To amend this the system could be extended to add a calculation to determine how skewed that bet is, compared to the likely value, or maximum value of the hand. Extending the system like this would add a dimension behaviour, and the system would be less predictable – the benefit of this is that it is quite easy to extend the system in such a way (just add more patterns to be checked) as well as this it is a more straight forward design and implementation, as long as the developer understands the underlying theory of the game behind it.

On the other hand, the system still retains of its biggest drawbacks – it will take a lot of time and resources to develop a system like this, because, there are a lot of different possible states for a game like poker, and whilst it is possible to generalise these states, to a degree, if one common scenario is missed out and the system defaults to a linear playstyle, then the system becomes very easily exploitable and has failed its requirements. It is quite hard to depend on the robustness of a system designed like this. In addition to this, it will require a lot of maintenance, because, it is unlikely that all the scenarios will be accounted for on with its launch, as a result the system would require constant updating to account for these scenarios, moreover, until it is all amended, the system will not have fulfilled its goal.

Lastly, to fulfil the requirement of personalities as a part of the system, this would involve even more work as the developer would have to design and implement several behaviours for the same system, based on one scenario – one for a passive personality, one for an aggressive personality etc. This makes this solution less viable, based on the time scope and resources of the project.

Main Points:

* Easily extendable
* Complicated design process.
* Simple implementation, but high implementation time.
* High risk design – hard to reason about robustness
* High Maintenance

**Combination of Current System and Statistical Analysis - Semi-Manual System**

An alternative solution would be similar to the current system, but with some of the fundamental concepts altered. For example, rather than hard-coding patterns for the system to check against, the system could instead use a combination of hard calculations, combined with statistical analysis to determine its behaviour.

For example, when determining the value of a hand, the system could still use a subroutine to determine the theoretical value of its own hand, and the probability of the value of the opponents hand relative to their bets, however, when determining its next move, it could use statistical analysis to determine the probability of the current state of the game leading to a win, and then basing its next action upon that, rather than a simple comparison, and pattern matching system as proposed by the current system.

This could be implemented in two different ways, for example, the system could use decision trees to predict future states of the game, and then based on the average value of those outcomes, and it would make its move. This is similar to the current system, however, rather than hard coding these states in, it would be more self-sufficient, by determining the value of its current position by itself. However, this has the potential to be a very inefficient and slow method, as there are so many states that the game could go into, it would take a long time for the system to analyse all of them and determine their value, especially early on in the game. Moreover, if the system took this approach every time, the decision of the system may stagnate as the value of future outcomes may be quite similar to each other, regardless of the move of the system in certain scenarios.

Another implementation of this analysis would be to have a record of previous games, either played by the system, a separate data store, or a combination, then use this data to determine the likelihood of the current value of the position of the system, based on data from previous games. The benefit of an implementation like this is that the system would be make a move more quickly, as long as the data is stored and can be queried in an efficient way. The drawback of this however, is that this system would require a large data store of previous games in order for this to be effective, or else anomalies in the data would have a large impact in the effectiveness of the systems behaviour. This implementation would only be viable if a large store of poker games and results could be found.

Main Points:

* Combination of current system and statistical analysis, in order to reduce linear behaviour of the system and to make the system more generally effective. Similar to last current system, with less maintenance required.
* Implementation of statistical side can be with either decision trees, with values calculated by the system, or from large data store about previous games.
* This system is more self-sufficient as it could improve with more games, making the system less predictable.
* Implementation for personalities is easier as threshold for behaviour (based on calculated value of current position) can be adjusted to emulate different personality behaviour.

**Neural Network / Machine Learning System (Pseudo-automatic)**

Another potential solution would be to model the problem as a neural network and use this model in conjunction with a learning algorithm to generate a system with learned behaviour; this could also be enhanced with use of hard programmed edge cases, or some extra analysis. I could implement this in two ways: either I could use a linear algebra library or as along as it is efficient in its operations I could design my own network and implement algorithms with more freedom to adjust it, alternatively I could use a library with prebuilt neural networks and then program it to fit the problem, which may be more reliable but the trade-off is that I have less freedom in the design and implementation.

This solution would involve modelling a game of poker as a neural network, with the condition of each card in the hand of the system, as well as the state of the pot and community cards as the features, or inputs, and the output would be a single neuron of how much to bet/raise (or if it is 0, or a low number, then fold). Poker is a game of incomplete information, which makes it hard to calculate accurately the value of the current position of the system, as well as using hard calculations to map patterns with the state of the game to linear behaviours, as a result, a neural network model with well-learned weights could be a better solution for producing an effective AI. The reason a neural network would be used, rather than just a classification or polynomial regression system is that it can more easily be used to implement a polynomial system, however, either of those could work, as there would not be a huge number of features in this system.

In conjunction with this would require a learning algorithm. There are a few options in this case, for example, if I could find a large data store of poker games, I could alter the system to optimise a supervised learning problem, and use an algorithm such as gradient descent, to try an learn weights which are most effective for a certain input state. This has the issue for potentially being a bit predictable, and is susceptible to the data (this entire solution is susceptible to the data, but this one more than others) if it is a record of games which are played extremely non-optimally, then the system will try to emulate this and play equally as bad.

An alternative would be an unsupervised problem, where the system would use something such as reinforcement learning or neuro-evolution algorithm to train the system based on games it has played. The issue with this is that, before the system is viable, it would have to play a large number of games to learn from, this would entail having to play a lot of games against a human user (unviable) or another AI. This to some degree solves the issue, of the requirement of a large data store of poker games, because the system, in this case, could be set up to play games against itself, and then adjust the weights of its network, based on which moves worked and which ones did not – if the number of games played is high enough, this should eliminate anomalous moves, or “cheese” strategies which results in a short term win against an AI, but which would be spotted with a human user – such as maybe all-in’ing every round, to intimidate the opponent into folding. Another issue arises in this case, of being stuck in local optima, to amend this the system can be initialised with many random weights, and then pick the highest effective weighting to apply to the system, or pre-programming some initial behaviour to bring the system up to a given level, rather than just a random level, and letting it train from there.

In relation to the issue of personalities, it would be simpler to implement that in this system, as the system could be retrained with either a random weight, or its most effective weighting from the default training, and change the reward calculation (for example, place a higher emphasis on short term winnings for an aggressive personality), then retrain the system, for each personality.

Lastly, this system would be the most self-sufficient out of all the proposed systems, because it each game it plays it would be able to add this new game to its training, resulting in a non-linear behaviour, as each batch of games would be used to adjust its behaviour. This would result in a less predictable system which requires less maintenance.

Main Points:

* Model game as a machine learning system with a few features – the low number of features means efficient processing and learning.
* Weights can be learned either supervised (if database of poker games can be found), or unsupervised (reinforcement learning / neuro-evolution algorithm).
* Personalities somewhat simpler to implement – as reward for the system can be adjusted in relation to the desired output, based on personality types (eg. Higher emphasis on short term winnings for more aggressive personalities)
* Self-sufficient as each game it plays can be used to change its behaviour – this also makes the system less linear.

**Other Potential Solutions:**

* The developer changes their design of the final product, to just being a multiplayer game – thus eliminating the need for an AI system entirely. Not viable, because this is the clients choice, not mine
* Buying a retail AI made for playing poker, and adapt it to fit the current system. This is unviable, because prices of Poker bots can be upwards of a thousand dollars – this is not a reasonable price range for the scope and requirements of this project.

### Proposed Solution

My proposed solution is going to be the Neural Network based solution. I have picked this one, because I feel that it would be the best for the incomplete nature of poker, for the reasons outlined above. Consequently, this means that the system would be effective against higher skill players, and it is easier to purposely limit a system one it has shown that it can perform better, than picking a less effective method and then trying to push past its optimal ceiling.

In addition to this is potentially the best solution to implement the most difficult, arguably, aspect of the system – the personalities. This can be done by changing the reward system (if I use a reinforcement learning algorithm, for example) to correspond to each of the personalities (detailed above). For the other solutions, I would have to adjust with their calculated output manually, and this increases the risk of the system being more linear than desired.

As well as this, for a development team this size, minimal maintenance is paramount, which is another reason why this solution is the most optimal. Due to the nature of the system being able to adapt with beyond the release. Moreover, this quality of utilising a datastore of games to adapt, means that I can use some of the aspects of the statistical analysis solution in conjunction with this solution; I can go further and even hard code some behaviours, if I feel they are fundamental enough, and that the system is not exhibiting them.

Overall, this solution is the most versatile, and adaptable solution, which makes it an optimal choice.

#### Proposed Solution DFD

**Level 0 DFD**

The level 0 DFD remains the same, as it is performing the same high level process, however, it will be doing it in a more optimal way.



****

#### DFD for Game System



#### ERD for Proposed Solution



#### Proposed System Neural Network Architecture



On each of the connections of the neural network, there will be a weighting applied , which will correspond to how much “weight” that neural connection has in the output of the network. Here the output will be a n integer, which will correspond to how much the network wants to bet, alternatively how much value the AI places on its current position, and if it is 0 or less than the minimum bet amount then this will correspond to the AI folding. These weightings will be generated using a form of training – most likely reinforcement, however, neural evolution will also be tested.

The activation function for each neuron will be a linear activation function. I have chosen this because I want the network to process the value of its current position, or potential future value of its next move, as a result, a linear activation function fits this need best – as the higher the value of its current position, the higher the bet should be.

This bet will also be processed through another function, to change the final bet, so that the bet can be adjusted for each of the personality types. For example, if the designated personality was more aggressive, it would have a different weighting on its current cards and the pot, compared to other personalities, however, the output bet would be changed to be higher than that of a passive player. An alternative to this could be to use a different activation function, for each of the different personalities.

### Prototyping and Critical Path

The most difficult part of this project will be developing a competitive base AI, which can play the game to a level which is suitable to the user. As a result this is the first part of the program which I will prototype. To do this I will first develop the neural network, and train it using a reinforcement learning algorithm, by having it play against another iteration of itself with different randomly generated weights which will also be improved with the same algorithm. After a designated number of iterations, I will play 5 games against it and determine if I can find any exploitable or predictable patterns, after this I will ask a friend who is more experienced in poker, to also play 5 games against it for the same goal. If this is not effective, I will attempt to develop a new set of weights, either my more iterations, or by using a different learning algorithm, in this case I would use a neural evolution algorithm to develop new weights, and then use the same process to test how effective the AI is.

After this the next most difficult part of the development will be either, to make the AI difficulty adjustable, or to develop the personality types for the AI. I will develop the system for the personality types first, and then use the same testing method to determine if the AI is still a viable opponent, I will also use the same testers for each game to see if its behaviour is different compared to previous games played against it – to see if its behaviour has changed. In addition to this, I will also have the client play same games against the AI to see if the different personality types I have put in play fit in with what he wants from a design perspective.

One this has been completed, I will develop the system to adjust the difficulty of the AI. I will develop this part, and then just use myself or another new player to Poker to determine if the AI is within a competitive difficulty. As it has already been tested from a baseline perspective against a more experienced poker player before this, it should already be competitive at the highest level this AI has been designed for, therefore I would only need to test it against lower skilled opponents. Essentially this is like maximising the playing level of the AI and then limiting it for more casual users, as this is easier than developing it for casual users and then attempting to increase its ceiling. In addition to this I will implement a system for the AI to detect if the difficulty is skewed for its opponent, and to adjust its difficulty automatically if it is. To test this I will set it to its maximum level, and then play against it and either make purposely bad moves, or just play normally, and see if it adjusts its difficulty level after several losses in a row.

The next most challenging part to develop will be the adaptation against particular users, although this is arguably more challenging to develop compared to than the difficulty adjustment, this is a less important aspect compared to that more fundamental objective. To develop this, I will have to develop the database which will store the games played, and the users played against. Once this has been set up, I will design a log in system for the user, and then play against the AI with a particular style in mind, to see if its behaviour changes, if it does then I can have confidence that it fills out this objective, to some degree.

After this, the bulk of the system should be prototyped and tested. Therefore, after this the full interface should be developed, tested and polished, so that the AI is ready to be deployed, without opportunity of being potentially exploited, due to an interface which provides too much access.

## Documented Design

SoftwareDevelopment

## Testing

## Evaluation

1. <https://archive.ics.uci.edu/ml/datasets/Poker+Hand> (Repository of 5 Card Draw Hands) [↑](#footnote-ref-1)
2. <https://en.wikipedia.org/wiki/Monte_Carlo_method> “Monte Carlo methods (or Monte Carlo experiments) are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. Their essential idea is using randomness to solve problems that might be deterministic in principle.” [↑](#footnote-ref-2)
3. <http://www2.cs.uregina.ca/~hilder/refereed_conference_proceedings/cig09.pdf> (No-Limit Texas Hold’em Poker Agents Created with Evolutionary Neural Networks – Garrett Nicolai and Robert J. Hilderman). [↑](#footnote-ref-3)
4. <http://www.numpy.org/> (Scientific Computing Library for Python) [↑](#footnote-ref-4)
5. <http://pybrain.org/> (Neural Network and Reinforcement Library for Python) [↑](#footnote-ref-5)
6. [www.pokerology.com/lessons/poker-playing-styles/](http://www.pokerology.com/lessons/poker-playing-styles/) (Article on poker playing styles) [↑](#footnote-ref-6)