Hybrid Agent for Fighting ICE Competition

Gary White
Trinity College Dublin
College Green
Dublin 2, Ireland
whiteg5@tcd.ie

ABSTRACT

In this paper, an alternative architecture is proposed for the Fighting ICE competition which combines two of the most popular previous architectures reflex and learning based as subsystems. In this paper I present a hybrid based architecture which combines two subsystems one reactive and one proactive and combines them into a single agent. A simple reflex agent is used to build in some expert knowledge and k nearest neighbour is used to predict attacks based on location. The sub-system is modelled as a vertically layered one pass architecture. This allows for the combination of different agent architectures at each layer and to choose the optimal action based on the perception system.

Keywords

Artificial Intelligence, Hybrid Agent, K-Nearest Neighbour, Reflex Agent

1. INTRODUCTION

Fighting games are one of the most popular types of video games and are usually played by two humans. In recent years there has been a large focus in the AI community on these fighting games as they provide an excellent platform to build new techniques which can work in real-time, one of those platforms is the Fighting Ice competition [3] which can be seen in Figure 1. A brief description of the environment is given in Table 1. Most of the previous entries to the Fighting ICE competition have been designed as simple reflex agents based on expert knowledge from the designers. Some learning and fuzzy logic techniques have been explored in the past to varying degrees of success [4] [5]. In this paper I present a hybrid model that attempts to use the best of rule and learning based techniques that have previously been researched.

The game is fully observable which is why reflex agents preform so well, one of the important rules in the game however is that there is a 0.25s delay from the simulation to the agents. In the game the smallest unit of time is a frame, which means that every action can be counted by this. For example the movement speed will be 10 pixels per frame, a light punch will hit the opponent after 8 frames and a complex skill must be input in 4 frames. One second is 60 frames so the response time is 16.66ms/frame. This means that our learning algorithms have to very efficient at predicting the opponents skills to ensure that we are able to defend against them in time.

Table 1: Description of environment

Environment	Observable	Deterministic	Episodic
Fighting AI	Fully	Stochastic	Episodic
Static	Discrete	Agents	
Semi	Discrete	Single	

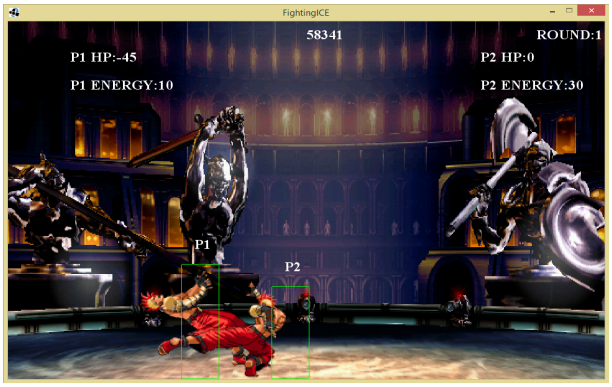

Figure 1: Fighting ICE Simulation

Reflex agents have been very successful in the 1C competition, where there is only 1 character so you can easily build a reflex agent to deal with most of the attacks that your opponent will use. There has been an effort by moving to the 3C competition where there are 3 characters each with different attributes, to encourage the development of learning based AI's as it is harder to make a list of rules for each character you could face.

One of the challenges of building an agent for the fighting game is the time constraint both in terms of the time you have to make each move 16.66ms and the amount of time you have to learn about the opponent 3 sixty second rounds. One of the reasons why learning agents have not been suc-

cessful in previous competitions is due to these time constraints. You have very little time to collect data from your opponent, and as you must play against each of the AI's in the competition your agent needs to be able to adapt. You have very little time to process information on your opponent and if you don't do it quickly enough then your agent will not have any input and will be defenceless.

There are a number of other rules about the competition however the most important are that you are not allowed to use multi-threading, class loading or reflection. The agents have a memory limit of 512MB (including the game) and if your agent exceeds this limit it will be disqualified. Your agent also only has 5 seconds to use as ready time to initialize. The winner is decided using formula (1), HP starts from 0 and decreases as your agent is hit.

$$\frac{Opponent\ Charachter's\ HP}{Plyaer\ Charachter's\ HP + Opponent\ Charachter's\ HP} \quad (1)$$

In this project I present the use of a hybrid agent which has the ability to make reflex decisions based on the opponents distance, energy and position i.e. whether on the ground or in the air. It also has the ability to predict opponents moves based on their previous moves in a given location, which allows for effective countermeasures to be used. The hybrid architecture allows for the modification of the learning agent to give more accurate results as we do not need it to predict all attacks as we have other rules we can use, so we can focus on achieving high accuracy. As can be seen in the next section a number of different classification techniques were explored before settling on the K-Nearest Neighbour Algorithm.

2. PROBLEM DESCRIPTION

In this section I describe the main problems that were addressed over the course of the project. Some of the problems are fundamental to the design of a successful agent such as what type of architecture such be used? Each subsection contains a summary which identifies the finding of my research and the technique that I would use in the implementation.

2.1 What architecture should be used when designing a Fighting AI?

This is one of the most fundamental problems addressed in the project as it effects the whole design of the agent. There are a number of different resources which describe the benefits of different architectures [6]. I consider a number of architectures some which are based on previous submissions to the competition and others which I thought would be a good to explore to see if they would be suitable for use in this competition.

2.1.1 Simple Reflex Agent

These agents select actions on the basis of the current percept, ignoring the rest of the percept history which makes them easy to design and surprisingly effective in fully observable situations. The agent function is based on the condition-action rule: if condition then action, which can be seen in Figure 2. This makes them easily programmable for expert rules such as if the agent has enough energy use the special attack.

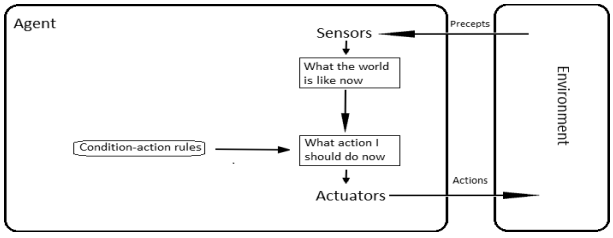

Figure 2: Simple Reflex Architecture

Simple reflex agents have the admirable property of being easy to implement, but they turn out to be of limited intelligence. The agent will work only if the correct decision can be made on the basis of only the current percept, that is only if the environment is fully observable. One of the major problems with the reflex agent is the lack of feedback in the system, if the opponent finds a flaw in the agent it will be able to exploit it and the agent won't be able to adapt.

Simple reflex agents have been widely used in the competition so far and have enjoyed a great deal of success as they are easily programmed with expert knowledge. They have been able to outperform much more sophisticated models by having a large amount of expert rules and always having a default action that they can use.

2.1.2 Learning Agent

Learning agents have an advantage over reflex agents by allowing the agent to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow. This is useful in the Fighting Ice competition as it allows agents to adapt to different opponents. One of the areas that learning based agents would be useful would be in predicting opponents attacks as they can use classification algorithms to predict what attack an opponent will use based on there position and energy.

Some learning based agents have been tried in the past in this competition but they have usually been too slow to adapt to the changes in new opponents. This makes them vulnerable in the first round to the simple reflex agents when they are still trying to learn from the opponent AI, as there opponent will already have a large amount of expert knowledge built in.

2.1.3 Utility-based Agent

It is possible to define a measure of how desirable a particular state is. This measure can be obtained through the use of a utility function which maps a state to a measure of the utility of the state. A more general performance measure should allow a comparison of different world states according to exactly how happy they would make the agent.

A rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes- that is, the agent expects to derive, on average, given the probabilities and utilities of each outcome. A utility-based agent has

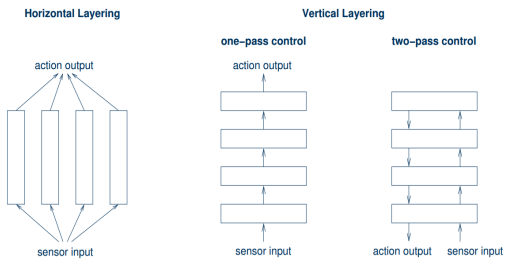

Figure 3: Hybrid Layers

to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning.

There has been no utility based agents in the competition so far, one of the reasons I think this is the case is that they can by difficult to design. The fighting simulation is complex and there are a lot of different moves and time considerations to take into account when designing a utility based system. I think that if one agent where to achieve success from this system then it would allow other to take the initial design and build on it as laying the foundations is quite tricky.

2.1.4 Hybrid Agent

While researching the what architecture should be used when designing a fighting ai I came across the hybrid agent architecture. The basic idea of the hybrid architecture made sense from what I had seen from my research into the other agent types that neither completely deliberative nor completely reactive architectures are suitable, so combine both perspectives into one architecture.

The most obvious approach is to construct an agent that exists of one (or more) reactive and one (or more) deliberative sub-components. The reactive sub-components would be capable to respond to world changes without any complex reasoning and decision-making. The deliberative sub-system would be responsible for abstract planning and decision-making using symbolic representations.

There are a number of different techniques for building a hybrid based agent but the most commonly used are layered approaches which can be seen in Figure 3. This can be by horizontal layering where all layers are connected to sensory input/action output and each layer produces an action, different suggestions have to be reconciled into one action. Another approach is vertical layering where only one layer is connected to sensors/effectors and filtering approach propagate intermediate decisions from one layer to another.

2.1.5 Conclusions

I have chosen to implement a Hybrid agent based on the research that was carried out in this section. As none of the previous entries to the competition had used this architecture I thought that it would make an interesting project to explore this architecture as an option for future agents. It also allowed for the research into a reactive and deliberative subsystem, and as the aim of the project was to

Guard type / Attack type	Stand guard	Crouch guard	Air guard
High	block	block	block
Middle	block	hit	block
Low	hit	block	hit
Throw	hit	hit	miss

Figure 4: Attack & Guard Moves

learn about the application of AI techniques I though that developing with this architecture would produce the most complete project.

The design combines two subsystems one reactive and one proactive and combines them into a single agent. The subsystem is modelled as a vertically layered one pass architecture which can be seen in Figure 3. This allows for the combination of different agent architectures at each layer and to choose the optimal action based on the perception system.

2.2 Can you predict an opponents attacks?

One of the most interesting questions that I tried to look at was the prediction of opponents attacks. An accurate prediction of an opponents attack gives a huge advantage to agent as moves can be either blocked or countered against as can be seen in Figure 4. Fortunately the topic of prediction has been explored a great deal and there is a large number of established techniques which are explored in the next section.

I decided to approach this problems as a classification problem where we would try and classify the opponents attack based on its position. There are many approaches to classification which can differ in complexity, speed and ease of implementation [7]. In the next section I describe some of the techniques that were looked at over the course of this project.

2.2.1 Bayesian Pattern Ranking

One of the first techniques that I though would be a suitable approach to predicting an opponents attack was to use Bayesian Pattern Ranking which had shown some promising results when being used for move prediction in other games such as GO [8]. One of the problems that I encountered for this approach would be that I would need a large record of expert games in order to train my system on, in the paper they use a collection of 181,000 expert games to train the system.

Go is a hugely popular game that is played by a large number of humans and AIs. This means that there are resources with a large number of expert games to train new agents on. This unfortunately isn't the case with the Fighting ICE competition as it was designed specifically for AI competition. The organisers release the .jar files for the previous years competitors which means that it is possible to create a database of your own for the best preforming agents in the previous years.

One of the other major obstacles for this approach is its

speed, the fighting ai competition is a lot different to Go, in the time you have to make moves. In the fighting game there is huge importance in being able to select a move as quickly as possible because if you don't enter a move you are left defenceless. There is no time limit on a traditional game of Go which means that it has time to preform analysis on the opponents moves.

2.2.2 N-Gram

The next technique that I tried to use to predict moves was N-Gram, which is a continuous sequence of n items from a given sequence which in this case would be a sequence of moves. The N-Gram model then gives a prediction of the next move based on the previous sequence, this has been used in a wide number of areas from natural language processing [1] to predicting web requests [9]. This technique should be excellent for predicting combination moves from the opponent which would appear as clear patterns which the agent would then be able to defend against.

As there are quite a lot of moves in the game so we would need a hash table to store the N-Gram data. This would take up quite a large amount of memory as you have to consider all possibilities and as the moves have been altered for each of the new characters it would take up three times as much data as the original competition. One of the ways to reduce memory consumption would be to treat every N-Gram entry below a certain probability threshold as the same rare event. For example it wouldn't matter if the probability sequence was 0.5% or 0.7% it would be enough to know it was under 1% since those types of events are very common it can save a lot of memory.

As there are a number of different AI's which take part in the competition the algorithm needs to be able to adapt to each new AI while playing them. This requires a special sort of N-Gram algorithm called hierarchical N-Gram, which uses N-Grams in increasing order in parallel. I was able to implement a basic version of the N-Gram algorithm but when trying to implement the hierarchical N-Gram I could not get it to function correctly in time which made my AI miss inputs which left it vulnerable to attacks.

2.2.3 K Nearest Neighbour

The k-nearest neighbour algorithm is amongst the simplest of all machine learning algorithms. There are three key elements of this approach: a set of labelled objects, e.g., a set of stored records, a distance or similarity metric to compute distance between objects, and the value of k, the number of nearest neighbours. To classify an unlabelled object, the distance of this object to the labelled objects is computed, its k-nearest neighbours are identified, and the class labels of these nearest neighbours are then used to determine the class label of the object.

An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common amongst its k nearest neighbours. K is a positive integer, typically small. If k = 1, then the object is simply assigned to the class of its nearest neighbour. In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids tied votes.

There are several key issues that affect the performance of kNN. One is the choice of k. If k is too small, then the result can be sensitive to noise points. On the other hand, if k is too large, then the neighbourhood may include too many points from other classes. Another issue is the approach to combining the class labels. The simplest method is to take a majority vote, but this can be a problem if the nearest neighbors vary widely in their distance and the closer neighbors more reliably indicate the class of the object. A more sophisticated approach, which is usually much less sensitive to the choice of k, weights each objects vote by its distance, where the weight factor is often taken to be the reciprocal of the squared.

KNN classifiers are lazy learners, that is, models are not built explicitly unlike eager learners (e.g., decision trees, SVM, etc.). Thus, building the model is cheap, but classifying unknown objects is relatively expensive since it requires the computation of the k-nearest neighbours of the object to be labelled. This, in general, requires computing the distance of the unlabelled object to all the objects in the labelled set, which can be expensive particularly for large training sets.

A number of techniques have been developed for efficient computation of k-nearest neighbour distance that make use of the structure in the data to avoid having to compute distance to all objects in the training set. These techniques, which are particularly applicable for low dimensional data, can help reduce the computational cost without affecting classification accuracy.

2.2.4 Summary

After conducting detailed research into the large amount of techniques which can be used for predicting I choose to develop the KNN algorithm further as it could easily be developed as it is one of the simplest algorithms. Despite its simplicity it has been shown to preform well in a number of situations. In particular, a well known result by [2] shows that the the error of the nearest neighbour rule is bounded above by twice the Bayes error under certain reasonable assumptions. Also, the error of the general kNN method asymptotically approaches that of the Bayes error and can be used to approximate it.

KNN is particularly well suited for multi-modal classes as well as applications in which an object can have many class labels. For example, for the assignment of functions to genes based on expression profiles, some researchers found that kNN outperformed SVM, which is a much more sophisticated classification scheme [2]. As the time constraint was quite a large factor in this project I decided that it would be best to get a simple model working first and then explore some of the other more complicated techniques.

2.3 What is the best character to choose?

As there is a choice of characters for this years competition I though it would be worth taking a look at each of the characters motion data which is available on the website. The motion data contains all the information about the character in a csv file. This can be used to do some analysis on the strengths and weaknesses of each of the characters as can be seen in Figure 5. We can see from this figure that most

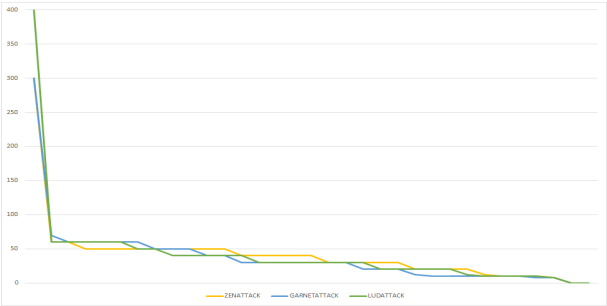

Figure 5: Character Attack Moves

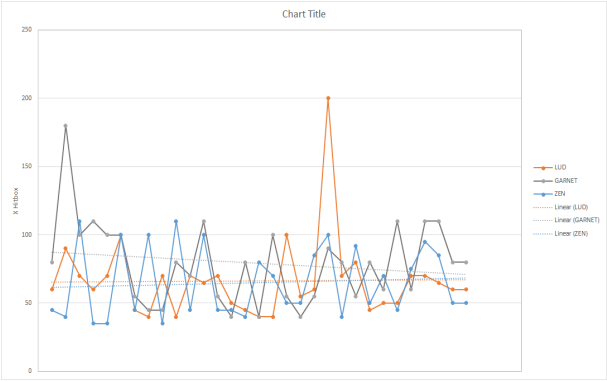

Figure 6: X Hitbox Size

of the attack moves are quite similar except for the special attack which is 400 for LUD and only 300 for GARNET and ZEN. Although the energy requirement so the LUD attack is 100 higher when the LUD attack is blocked it still does 200 damage whereas the other two only do 100 damage.

Most of the other attacks use a similar amount of energy and do a similar amount of damage as can be seen by Figure 5. I decided to look at the size of the hitbox for each of the moves to see if this would show any differences between the characters. Figure 6 shows the difference in hit distance in the horizontal axis for each of the characters, the attacks are ordered by attack damage with the most powerful on the left. We can see by the trend-lines that Garnet has the best hit distance for the most powerful attacks while Lud and Zen have similar hit boxes for all attacks.

From my research into the different strengths and weaknesses I think that LUD is slightly better than the other two because of the larger attack damage for the special attack. Although this attacks costs more energy to use I think that it is worthwhile as is does double the damage of the other special attacks when blocked. The start-up time of attacks is also a key feature when choosing a character as whoever hits the opponent first can cause a flinch. From my research into the startup time however I found no major differences between the characters. I don't think that the advantage is so great that the character data would need to be patched or adjusted, I think that it adds another interesting aspect to consider when designing the agent and specifying your rules and attacks.

3. IMPLEMENTATION

In this section I discuss how the methods that were researched in the previous section were implemented. The full source code is submitted along with this report for further detail on the exact implementation but this section gives a good high level description of how the system works. The prediction agent is based on the description in [4] for the 1C competition.

3.1 Can you predict an opponents attacks?

3.1.1 Collecting Data

To predict an opponents attack it is important to get as much information as possible about the other agent. This means being able to collect data from the other agent while playing against it. The first layer of the hybrid system is concerned with collecting data from the opponent before it decides what attack or defence to use.

For each attack we want to classify it by the distance between the AI and by the position of the agents i.e. whether it is ground-ground, ground-air or some other combination. This makes the classification of the attacks a lot easier as we only have to classify in one dimension. I had initially considered incorporating distance, position and energy into KNN but it is much harder to try and classify data accurately as the dimensions increase.

3.1.2 Classification

After collecting all the data we still have to classify it using K Nearest Neighbours. Even though KNN is one technique its performance can vary widely depending on what parameters are used. The choice of K and the distance threshold will have a large impact on the performance of the technique. This is why I decided to spend some time experimenting with different values, to see what combination would give the best performance which can be seen in the next subsection.

The actual KNN algorithm is quite simple we identify the K Nearest Neighbours using the Euclidean distance. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

A drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number. One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point.

3.1.2.1 JavaML. Is a library that provides a number of machine learning techniques such as data manipulation, clustering, feature selection and classification. This makes it easy to experiment with the results obtained from sample fights and to try and find the most effective parameters for the algorithm by using a range of performance classifiers.

Table 2: Investigating the impact of K

K Value	Correct	Wrong	Percentage
1	5173	17223	23.09%
3	6078	16318	27.13%
5	7926	14470	35.39%
7	8578	13818	38.30%
9	9120	13276	40.72%
11	8435	13961	37.66%

I tried to implement this approach of performance classification during the fight but found that most of the classifiers were too slow to be used in the time constraints needed and would often take over a minute to run through the large collection of data. There are a wide range of performance measures which can be used to test classifiers such as precision, recall, and accuracy. It is also possible to take a broader view and look at the incorrect and correct predictions and compare them directly.

The results from my experimentation with the K value can be seen in Table 2 where I used a number of simulations a previous entries into the competition. From this experimentation we can see clearly that we get much better predictions when using higher values of K. We get a maximum value of K as 40.72% for K equal to 9. These values serve as a good indication of what will happen but there is still a certain unpredictability when facing each new opponent. From this research I was able to conclude that I should use high values of K to try and achieve the most accurate predictions.

3.2 Reflex Agent

In order build a successful hybrid agent we must use the reflex agent to take care of the cases where the classification agent doesn't have enough samples to make an accurate prediction about what the move should be. We can also improve the agent by putting in some common sense rules that would be hard for the learning agent to implement. This shows the advantage of working with a hybrid architecture which makes the best of all the resources it has at its disposal. In the next sections I describe how I divided up the rules into different layers so that the hybrid agent could be easily tested and both agents could be developed simultaneously.

3.2.1 Special Attacks & Combo Moves

Special attacks are hugely important in the game as they do so much more damage than the regular attacks and use so much energy that it is important to make sure that they hit. To put it in perspective the character LUD's special attack does 400 damage and uses 400 energy even if the attack is blocked it does 200 damage, the next highest attack does 60 damage and uses no energy so we can see that the either hitting or missing a special attack can be a large turning point in the game.

As the special attacks are so important I decided to place them on the first layer of the vertically layered architecture which means that they are the first choice move if they are applicable. This makes sense as the energy is reset after every round so if your agent has enough energy to preform a special attack before the round ends this should be its first choice before predicting what the opponent attack might be.

It is also important not to waste the special attack by missing the opponent as even if the opponent blocks the move it will do 200 damage. By looking at the motion data for our chosen character LUD we can see the attack hit box which shows how we could miss the opponent when the opponent is in the air or has been knocked down. By checking for both of these cases we can ensure that the agent will always hit his opponent.

There are other skill moves that use energy other than the major attack, such as the slide. The agent switches between the slide attack and another attack that uses no energy based on the distance between the two agents and a skillcounter which is used to stop the agent becoming predictable. There are also a number of combo moves implemented which are executed based on the previous moves. This allows for a number of highly effective combination moves especially for air attacks.

3.2.2 Close Combat

Close combat is an important aspect of the game which is why I felt that it deserved its own layer. The reason that is it different to the other layers is that all the attacks will hit so it becomes a matter of choosing the right attack, rather than one that will hit the opponent. It is also important that the opponent will not be able to predict what attacks we will use in close combat, this is why I use the punchcounter variable to switch between the attacks if on is being used too much.

I have two distance measures for close combat if we are very close then the agent uses punch as it has a quick startup time which means that it will be able to hit the opponent quickly. If we is outside this range and the opponent is in the air then we try to counter by using a throw. If we were hit by a throw in a previous attack then we try and counter our opponent by using "STAND_D_DF_FB"

3.2.3 Defend Tactics

If we have not predicted the opponents attack and are not in a close combat situation then we can try the defend tactics layer. In this layer we try to defend against 4 different types of attack (high, low, middle, low, throw). Although not as effective as predicting because we don't get the chance to counter against the move it can still be effective and reduce the amount of damage that the opponents attack is capable of doing.

For each of the defending moves we use the opponents attack type which tells us what type of attack the opponent is doing (high, low, middle, low, throw). This is a very useful method as it allows us defend against a number of similar attacks at the same time and use the results from Figure 4 to try and defend.

3.2.4 Default Actions

The purpose of having this state is to ensure that we always have at least some default move that we can do. The agent only reaches this stage if is has not hit any of the other rules or been able to accurately predict the opponents

Table 3: Non-graphical simulation Results

Agent	Round 1	Round 2	Round 3	Average
T3C	500	666	1000	722
ATTeam2	250	500	751	501
DragonKing3C	707	670	320	565

Table 4: Graphical simulation Results

Agent	Round 1	Round 2	Round 3	Average
T3C	418	757	726	634
ATTeam2	478	415	518	470
DragonKing3C	447	493	578	506

attack. This is the last stage before the agent would be left defenceless so we ensure that we have one default move. Even though it is the last stage we can check a few rules before deciding on an action.

The main strategy of this layer is to get into a position where the agent is able to attack, we measure the opponents and our x positions and try to move closer to our opponent. The agent is aggressive and tries to be the first to make an attack.

4. RESULTS

In this section I go through some of the results that were observed through simulations with the agent. I used the graphical simulator to test the agent due to inconsistencies with the non-graphical simulator. In a later section I give an assessment of the AI techniques, this section focuses purely on the results that were observed from the questions that were posed in the previous sections.

4.1 Non-graphical simulation

I initially tested and developed my agent using the non-graphical simulator that is available on the competition Facebook page. I simulated 100 games against the top 3 agents from last years competition T3C, ATTeam2 and DragonKing the results can be seen in Table 3. The results however are very misleading as the non graphical simulation didn't work correctly and was giving me misleading results.

For example in the fight against the winner from last year my agent achieved a perfect score in the final round which is almost impossible to achieve. I kept getting the exact same results when repeating the experiment which led me to believe that the simulator was making some approximation of what might happen in the graphical simulation. It is clear that these results are misleading and were only included in the report to show the error in the non-graphical simulator which is described in more detail in the difficulties section.

4.2 Graphical simulation

After the results from trying the non-graphical simulator I decided that I would have to test my agent on the graphical simulation. This is a lot more time consuming than the non graphical simulation where each game would only take a couple of seconds, when in the graphical simulation you would have to wait the full three minutes for each game to complete to get the results. I was only able to use 10 games agents each agent due to time considerations but this still gives a good estimation of the fight.

My methodology was the same as the other simulation where I decided to simulate my agent against the top three from the previous years competition. We can see from Table 4 the difference in results, my agent preforms very well against the top three agents from last year and is able to beat two of them. We can see from Figures 7 - 9 the pattern that most of the games took. The black line is the winning point as it is the average of 500 so whoever is above this wins the round. The agent lost the first round in every game but managed to come back and win in two out of three.

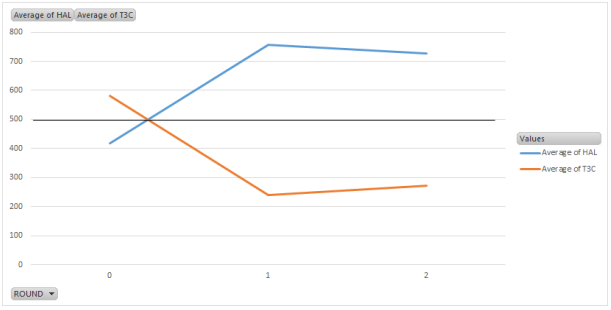

Figure 7: HAL vs T3C

4.3 What is the most effective architecture for designing a fighting AI?

It is hard to give a definitive answer to this question as I can only speak for the techniques that I have explored over the course of this project. From my experience it is clear that simple reflex agents are the best architecture to begin with as it is easy to design and gave give relatively good results quickly. The are also very intuitive to design and allow you to explore a number of different combination moves which can be really effective in the game.

From my experience with learning agents they are a lot more difficult to build and require much more effort to design well. In this case I looked at the KNN algorithm which is considered one of the easiest to implement, but I still found it tricky to achieve good results using just the learning agent. I was only able to give a brief period of time to researching the utility based agent, but it does seem like it would be able to work as we have an effective utility in HP.

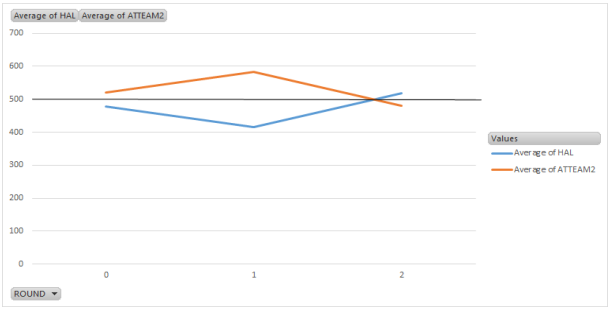

Figure 8: HAL vs ATTeam2

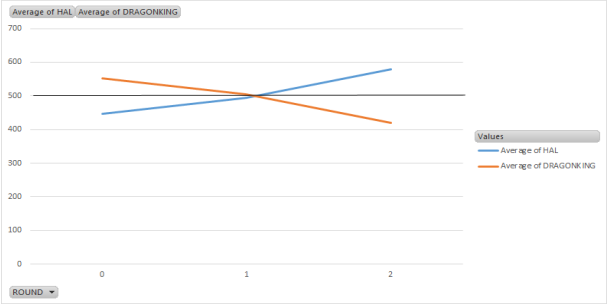

Figure 9: HAL vs DragonKing3C

I think that the hybrid agent described in this paper is one of the most effective architectures to design an agent for this competition. It combines the best aspects of the two agents that it uses the simplicity and effectiveness of the reflex agent and the ability of the learning agent to adjust to the opponent. Although in the time-line of this project I wasn't able to develop some of the more sophisticated learning architectures, I think that they could be effectively implemented in this system with a reflex agent to make sure that we get the best of both systems.

4.4 Can you learn opponents attack?

This was one of the central question of my research into this game and after some promising results using the results from simulations with the best agents from last year, when I actually implemented KNN the results were not as good and it took a lot of consideration and experimentation to find the correct parameters to use for the competition. I think that the final implementation was good and was quite successful at predicting the moves of each of the opponents it was tested against.

The best answer to this question is that it depends on what type of agent you are facing, if you are facing a reflex based agent then after recording some information about the agent you should be able to predict its moves. If however you are playing against an agent that chooses random attacks it is going to be impossible to try an predict the attacks.

I tried to focus KNN to only give highly accurate results by having a high value for K and a low distance threshold. This means that there has to be a large enough sample of moves from the opponent before we start using the results predicted by KNN. I also think that the system needs to make use of some performance classifiers and if they detect that the opponent is behaving in a random way then it would know to not try and classify the results.

4.5 What is the best character to choose?

After my initial research into this question by looking at the motion data associated with each character I had concluded that LUD was a slightly better character than the other two due to its more powerful special attacks. After carrying out a number of different simulations I was able to get the best performance when using LUD.

When designing an agent choosing the character is an important part of that decision, some characters can suit different styles of play. I think that LUD suited my style of play which is why I got the best results with that character. From the previous year entries there is a good mix of different agents so I think the answer is that the best character is the one that suits the design of your agent and there is no definitive answer.

After playing against all of the characters however I would build attack and defence rules based on the other persons character due to the difference in moves. There was some cases when the opponent did not use LUD the character that I built my system with that my agent would misjudge what attack the opponents was doing. To achieve an even better agent this would be an aspect worth looking at.

5. DIFFICULTIES

Over the course of developing my agent there were some difficulties faced, in this section I describe the main problems that I had over the course of the project and suggest some solutions for how the competition might be improved.

5.1 Graphical Fighting ICE Simulator

One of the major problems with the graphical simulator is that it is very slow to load all of the images needed for the simulation (40s) which means that it can be very time consuming to try out new features that you are developing in your agent. One of the other major problems that the graphical simulator has is that there is no option to speed up the simulation. This means that you have to wait the full three minutes for each game to complete. This makes it very difficult to build up a large collection of previous games to use for learning.

There is also quite a large amount of uncertainty in the graphical simulator and I often found that I needed 4-5 games to get a clear idea about how my agent was preforming. I recognise that this could be down to the unpredictability that is inherent in each fight but it did seem like there was a large variation between games which could often trick you into thinking some feature that you have added is a lot better than it actually is and having to wait 12-15 minutes to be sure slows down development.

5.2 Non-Graphical Fighting ICE Simulator

After some requests from the previous competition the developers released a beta version of a non-graphical simulator which could simulate a large amount of flights in one call from the command line on their Facebook page. I found however that the system was totally unreliable and would often predict the opposite of what would happen in the graphical simulation.

It also had a problem that if you decided to simulate 50 games between two different AIs then it would give the same answer 50 times. I would expect there to be similarities between the games but to have all the games with the exact same score suggests that they weren't introducing enough randomness into each game and where instead just following the same pattern of moves each time. This is something that really need to be fixed as it would have been a great resource to quickly develop new techniques and be able to simulate them on a large number of games. This would have made the whole development process a lot quicker and more agile.

5.3 Error Messages

One of the most frustrating aspect of developing the agent was the lack of clear error messages when the system would not work. It was especially frustrating after having waited 40s for the simulation to load to only to find that your AI is not producing any input and there are no error messages. If your AI caused the simulation to crash it would just give some general error about an exception in a thread and not give you a clear indication of what the real problem was, in other situations it would not give any error messages and the agent would just stand there and not perform any moves.

This led to a lot of wasted time searching for bugs, I appreciate that when the development team were creating the simulator they were probably concerned with other aspects of the game such as making sure the simulation has the correct timing and is able to simulate the fight in an entertaining manner. I would propose making the non-graphical simulator the development simulator which would have better error message generation which would allow for a quicker development cycle.

6. ASSESSMENT OF AI TECHNIQUES

In this section I describe the the effectiveness of the AI techniques that were explored throughout the course and the architecture that was used to implement them.

6.1 K Nearest Neighbour Classification

This was very effective in predicting the opponents attacks and allowed me to build a special prediction layer which could anticipate what attacks the opponent would do. As I also had some rules to fall back on I could increase the K value and narrow the threshold distance to make sure that I had a really accurate prediction. I was then able to implement specialist rules which were able to counter all the predicted attacks.

Experimenting with the JavaML library was really useful as it allowed me to quickly test a number of different classification techniques without have to develop them from scratch and to test their performance using advanced performance measures. This gave me a good idea of what the best parameters would be for KNN when I had to implement it in the agent.

6.2 Expert Rules

The expert rules that were built into the system were very effective as there are some rules that are obvious after playing the game for a while such as knowing when to use your special attacks. Dividing the rules into different layers was an effective technique as it allowed the each of the rules to be tested independently and to think of the different situations that might need expert rules such as when using special moves, in close combat or to have some backup defence moves in case the agent is not able to predict the opponents attack.

The previous agent in the competition were a great resource for ideas to build rules from as each of the previous teams gave a presentation on their agent which is on the website. This would usually involve which moves they had found useful and how they had designed there rule base for example as a state machine.

6.3 Hybrid Architecture

I think that the hybrid architecture was extremely successful in this case as it got the best out of both of the agent subsystems that were used. The learning agent was able to make more accurate predictions because we altered the parameters to give higher precision even at the expense of recall. I focused the system on achieving a large amount of true positives, even if that meant having a few more false negatives.

I think that the vertical layer approach worked really well as it allowed the rules to be modularised which meant that they could be tested and developed separately. This made is easy to find any rules that weren't working as it would be clear what layer they would be in. I would have liked to have experimented with the horizontal layering technique which although more complex might possibly be able to offer improved performance over the one-pass vertical control. It would also be interesting to experiment with the two pass control vertical system where each of the rules would be checked and then we could decide on the best one to use instead of the one that hits first. This seems like it would be able to offer more control to our agent which should improve performance.

7. CONCLUSION

I think this has been an excellent project to explore a number of the latest techniques for AI design and to try and implement them as well. I though that the project was challenging but was also rewarding to see an agent that I had designed be able to preform well against agents that have competed in the competition before.

As the project was developed on such a limited time-line there are a number are a number of other techniques and ideas that I would like to explore as future work to the current project with a view to submitting an agent to the next competition. I think that the organisers really need to fix the non-graphical simulator as it makes it much more difficult to develop complicated agents when it takes so long to test them on the graphical simulator.

I would have liked to explore some search techniques such as Q* and SARSA which could be used to search through all available moves and try and find the one with the maximum attack damage. From some brief research into these techniques they seem like they would be well suited to designing an agent for this competition. It might also be possible to just add them as another layer to our hybrid architecture.

In conclusion I think that the project has been extremely successful in exploring a wide variety of AI techniques and taking each into consideration when designing the solution. I feel that the implementation could have been improved by implementing a different prediction technique but under the time pressure it was only possible to implement the simplest technique (which was still very effective). Some feedback for further iterations of the course which would be to start the final project earlier. This would allow for interaction with the teaching assistants to solve any of the initial problems with getting the game set up which can take up a lot of time and feedback on the initial techniques being explored.

8. REFERENCES

[1] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, Dec. 1992.

[2] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor*, 13(1):21–27, Sept. 2006.

[3] L. H. N. S. M. Y. L. Feiyu Lu, Kaito Yamamoto and R. Thawonmas. Fighting game artificial intelligence competition platform. *Proc. of the 2nd IEEE Global Conference on Consumer Electronics (GCCE 2013)*, 2013.

[4] C. Y. C. Kaito Yamamoto, Syunsuke Mizuno and R. Thawonmas. Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator. *Proc. of 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014)*, 2014.

[5] H. Park and K. Kim. Learning to play fighting game using massive play data. *Proc. of 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014)*, 2014.

[6] S. Russell, P. Norvig, and A. Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

[7] A. Statnikov, C. F. Aliferis, I. Tsamardinos, D. Hardin, and S. Levy. A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis. *Bioinformatics*, 21(5):631–643, 2005.

[8] D. Stern, R. Herbrich, and T. Graepel. Bayesian pattern ranking for move prediction in the game of go. In *Proceedings of the 23rd international conference on Machine learning*, pages 873–880. ACM, 2006.

[9] Z. Su, Q. Yang, Y. Lu, and H. Zhang. Whatnext: a prediction system for web requests using n-gram sequence models. In *Web Information Systems Engineering, 2000. Proceedings of the First International Conference on*, volume 1, pages 214–221 vol.1, 2000.