BDI agent with Machine Learning for 2kbotprize

Angela Kravcevich
Trinity College Dublin
College Green, Dublin 2

ABSTRACT
This paper provides an overview of intelligent thinking during decision making of an agent in a First Person Shooter game called Unreal Tournament. The AI was developed using a Pogamut module called Gamebots 2004 in Java using SQLite as a memory component. The goal of the AI is to trick a human into thinking it is another human player.

The main focus of implementation was the Belief Desire Intention architecture that was modeled after interview surveys conducted on volunteer participats. Later, machine learning was added to provide a portion of human unpredictability based on the Multi-Armed Bandit problem.

Categories and Subject Descriptors
H.4 [**Autonomous Agents and Reinforcement Learning**]: BDI, n-armed bandits

General Terms
Algorithm, Design

1. INTRODUCTION
In most First Person Shooter games, the Artificial Intelligence agent plays an important part in the entertainment aspect of the game. Most face the issues of being predictable, hence easy to counter and overal reduce enjoyment of playing versus non human oponents. The 2k botprize is a competition that challenges programmers to develop an AI for Unreal Tournament 2004 that can fool an opponent into thinking it is another human player. The competition is sponsored by 2K games, the developers of UT2004. The agents are measured using a Turring test to determine if they exceed the human likeness threshold.

To implement AI with the most human likeness, factors such as combat, navigation and decision making need to be concidered. The main focus of this paper is to develop intelligent decision making process for an agent.

When observing the thought process of an average human player in an FPS(first person shooter) game, it is noticeable that most follow a set of common behaviors. Hence an idea came to life in an attempt to capture this behaviour using BDI architecture. However, by outlining such specific responses to particular situations we are making the agent structured predictable. To enhance this and to replicate human unpredictability and spontaneous decision making in FPS games, an additional machine learning module was implemented. Implementation of each component and the thought process are discussed in following sections of the paper.

2. INFORMATION GATHERING
To insure appropriate responses are modeled after a human player, several volunteers where gathered to participate in a quick interview. Questions where presented in a way to create a specific situation for the player and enquire about their response reaction to this situation. Sample questions that where asked:

- "Upon noticing another player, with current amount of health being low and no health pack available, what would you do?"
- "At the start of a game on an unknown map, what is the first thing you do?"
- "Considering all weapons are available in your inventory, what weapon would you use for a close combat engagement?"

Similar approach was taken in [Norling and Sonenberg 2004] where different types of FPS players where gathered and interviewed in an attempt to recreate their play styles. Additional 2 sample interview papers are included with this report to demonstrate such survey responses. The questions attempt to record all possible situation responses. However, if an unknown situation occurs then the AI should be able to adapt.

3. DEVELOPMENT ENVIRONMENT
The agent was developed using Pogamut plugin for netbeans and tested on a local server for UT2004.

3.1 Unreal Tournament 2004

UT2004 is an interactive, multi-player first person shooter game where multiple combatants compete in arena style set up. Players are provided choice in various maps, weapons and game types. 2K bot prize requires an implementation of an AI for a "Death Match" type game where the goal is to gain most points for killing other players. [UT2 2014]

3.2 Pogamut

Pogamut is a Java middleware that enables controlling virtual agents in multiple environments provided by game engines. Its main objective is to simplify the physical part of agent creation, hence why most of the modules such as pathfinding, navigation and combat are provided for dvelopers, so that they can focus on more entertaining development. [Pog 2014]

4. BELIEF DESIRE INTENTION ARCHITECTURE

Much work over the years has gone into study of computational agents that are capable of rational [Russell 1997] [Umarov and Mozgovoy 2012] [Rueda et al. 2002] The one I will primarily focus on is BDI Architecture[Norling and Sonenberg 2004], in essence it provides a mechanism for separating option generation, deliberation and action selection in a comprehensible manner. Every agent is modeled to have:

- Beliefs that represent the agent's knowledge of the world. Gamebots 2004 allows the agent to have access to list of players visible, enemy weapon, current health and inventory as well as locations of items on the map which would be known by human player through play time.
- Desires represent the agents goals and motivations. The agents purpose is to accomplish these goals while receiving the best rewards. Since the game is Death Match, the goal of the AI is to survive and kill the most enemies.
- Intentions are the current actions or a sequence of actions that the AI is performing, such as swapping weapons, navigating to an item or shooting an enemy.

```
initializeState();
repeat
    options := optionGenerator(eventQueue);
    selectedOptions := deliverate(options);
    updateIntentions(selectedOptions);
    execute();
    getNewExternalEvents();
    dropSuccessfuAttitudes();
    dropImpossibleAttitudes();
end repeat
```

Figure 1: BDI Algorithm.

There are multiple existing software languages for BDI implementation such as GOAL and JACK that have already been used for 2K botprize competition [Hindriks et al. 2011] [Norling and Sonenberg 2004]. Instead, I am implementing the BDI algorithm by extending an existing "Hunter" bot [Hun 2014], that was developed as part of demonstration for Gamebot 2004. Since I am only focusing on the decision making, the agent will use default combat and navigation modules provided.

The algorithm that demonstrates the main logic component of the agent is presented in Figure 1.

At the beginning of every cycle, the option generator reads the event queue and returns a list of options. This means that the agent looks at current items on the map and generates an array of navigation options, he checks weaponry for current weapons and their ammo count, whether he can see enemies around etc. All of these options are stored in a single object and passed to the deliberator for selection. The deliberator module narrows down the variety of choices depending on corresponding circumstances, for example: if low health and enemies nearby then the agent should retreat otherwise engage. Using the following decision making process, a location on the map will be selected that is either navigating the agent closer of further from the enemy. Once the exact actions are selected, the update-intentions method sends this information to specific modules of the AI, whether it is the memory, combat, navigation or weaponry. Over time, events will trigger the agents knowledge and hence provide different actions. [Norling and Sonenberg 2004]

5. IMPLEMENTATION

After gathering analysis from interviews, a setup of the baseline algorithm is created outlining importance of individual actions. What this means is an order of priority between actions is set up. For example, during item collection the order of priority is: armour, health, weapons, ammo. Hence, during option deliberation stage, these priorities will be taken into concideration. Similar idea goes to weapon selection and combat engagement. The initial algorithm implemented is shown in Figure 2.

Once the main priority algorithm is set up, the whole process is split into 3 separate stages based on the BDI algorithm:

- The first module gathers information about the current state of the world. Important events are: if an enemy is near by, how much health the player has, how much ammo and what weapon are in the agents inventory. This outlines the agents beliefs.
- The deliverator then examines all the gathered relevant knowledge and narrows down possible actions. If there are several items that are worth to get on the map, then the AI will randomly select one or will check the priority chart.
- Once the options are narrowed down, the update intention model comes into effect. This sends information to individual components such as the weaponry module to select a particular weapon based on the situation, or to tell the navigator what item to go to, or where to aim the shots. Since none of these where implemented by me and are the defaults built by Pogamut community, this demonstrates how easily new modules can be added to this architecture.

```
if !moving then
    reset();

weapons := weaponry.getWeapons();
maxPriority := currentWeapon(weapons);
while !bestWeaponFound(maxPriority) do
    maxPriority := weapons.next();
weaponry.selectWeapon();

if seeEnemies() then
    if compareWeapons() || chechHealth()
        backOff();
    else
        engage();

if exploring() then
    if armorExists() then
        gotoArmor();
    if medkitExists && checkHealth() then
        gotoMedpack();
    if betterWeaponExists() then
        gotoWeapon();
    if ammoExists() then
        gotoAmmo();

    selectRandom();
```

Figure 2: Basic Agent Algorithm

Once the BDI architecture is in place, an SQLite database was set up with a connection to the agent. This will act as a memory component. Currently only one table exists which represents the health parameter for machine learning which will be described in Section 6 on Reinforced Learning. The table consists of 3 columns all containing integer values with the health parameter being the primary key. Insert is applied to the table for every multiple of 5 health untill 200 which is the maximum. Armor was not included in this calculation as it is subtracted differently when damage is taken. For every row in the table a default of 1 win and 1 loss was added to create the 50% chance, talked about in Section 7 on Training. [Cothran 2014]

When the agent is loaded a connection to the memory unit is set up. During the gather information module when an enemy is spotted, the health parameter is recorded and a Select applied to the database, returning the loss and win ratios. These values are used to determine the chance of survival during an engagement. Event triggers are setup for the agent to take account of when the player or enemy dies. Once this happens the database loss/win counters are updated with respect to the situation.

For particular tactics such as comanding an agent to back off from an engagement while shooting or to pursue, individual methods where created based on the modified Hunter bot. Certain unintentional tactics are ovserved during the experimentation process. If an agent ha chosen to run away from an enemy while shooting and has walked into a dead end, the repeated confusion of the AI causes him to dodge in and out of a corner while shooting, making the overal action very human like. This is due to the agent having sight of the enemy and choosing to run away and with no where to go he has to go back the other way, hence causing a repeated action. In result the agent will either win the engagement or lose.

6. REINFORCED LEARNING

The goal of reinforced learning is to provide an agent with a set of situations and evaluating its choice of actions to maximize numerical reward over time. It is defined not by characterizing learning methods, but by characterizing a learning problem. One such problem is the "N-Armed Bandit Problem", so named by analogy to a slot machine where the 'n' represent the number of levers. The goal is through repeated plays of the slot machines to maximize the monetary reward by concentrating plays on the best lever. Each plays contributes to the decision making process of the next selection. Hence knowing the previous reward from each slot machine, one can develop a method using a greedy parameter that aids in selection of the next slot machine based on current knowledege. [Sutton and Barto 1998]

I am going to focus on a similar thought process of my reinforced learning implementation. To add an unpredictable selection process that is based on previous gathered knowledge, a memory unit is added to store such encounters. For purposes of implementation and testing, only a single parameter was focused on - players current health. The memory unit stores health in multiple of fives and their corresponding number wins and losses at that health. Initially all 200 instances where recorded for every one increment of health, this proven to be inefficient. Most weapons in the game cause a large amount of damage, hence having health in increments of one did not make sence. The smaller the increments, the more iterations it is required to reach optimal machine learning values that create sensible results.

An Action-Value Method can be applied to determine an estimated reward for the current health. When an agent encounters an enemy, he remembers the current health he has. Using the health parameter on the memory unit to predict a possible outcome of the engagement. This is done by computing a win percentage using the number of losses and wins the agent has from previous similar situations. If the win percentage is less than 50%, then the agent will most likely chose to back off rather than pursue. Comparing this to Action-Value method, where the desired action is to engage the enemy, t representing the current health of the player. The percentage computation is shown in Equation 1. The code implemented for $checkHealth()$ is shown in Figure 3.

$$Q_t(a) = (wins_t/(wins_t + losses_t)) * 100 \quad (1)$$

$Q_t(a)$ gives the percent chance to have a successful engagement. To create element of unpredictability, a random number is selected and compared to the win rate. If the value is within the win range, then the agent will choose to engage the enemy, otherwise the agent will back off. During the engagement, an event will trigger if the player or the enemy dies. Such an event increments the loss or win counter in the

```
wins := selectWin(engagementHealth);
losses := selectLosses(engagementHealth);
percent := wins / (losses + wins) * 100;
randomValue := randomDouble() * 100;
if percent < randomValue then
    return false
else
    return true
```

Figure 3: checkHealth() method that is utilised in Figure 2 earlier

memory unit, for that particular health. To insure accurate masurements of the health parameter, a pursuit timer was added. When the enemy is running away and is no longer in sight, in order for the health to not be reset when the enemy is spotted again, the timer allows a certain amount of leeway. The leeway code is shown in Figure 4

```
if seeEnemies() && firstEncounter then
    pursuitTimer := 0;
    engagementHealth := round(health, 5);
    firstEncounter := false;
else
    pursuitTimer++;
     if pursuitTimer > 30 then
        firstEncounter = true;
```

Figure 4: Pursuit Timer leeway to account for vision loss of target

Due to time limitation, health is the only parameter that was implemented for machine learning as a proof of concept. Similar idea can be applied to weapon efficiency, distance between players upon engagement, ammo based on accuracy etc. Most of the knowledge variables from the BDI can be applied to the memory unit to generate the unpredictability and variety factor. The mroe machine learning parameters are added to the agent, the more unpredictable and human like the player will seem.

7. TRAINING

During the implementation process, the memory unit was initialised with 50% chance of win and loss for every multiple of 5 health. This is to insure a purely random action selection. Over time, more reward values will be recorded to insure a more properly estimated reward. For testing the memory unit, an hour long experiment was performed where a human opponent was put against the agent. Offline training would be more effective, however due to the simplicity of the database that was used, multiple database writes where not allowed. A human player would also provide more accurate values in regards to human vs AI skill balance.

Figure 5 demonstrates the result after an hour training session with the memory unit. Only the first 100 health parameters where included, since the agent respawns with 100 health and there where little amount of health boosters on the test map to increase the health above 100, hence the results where negligible.

#	Value	Win	Loss
1	5	3	23
2	10	2	19
3	15	4	20
4	20	5	15
5	25	6	32
6	30	2	14
7	35	5	13
8	40	9	12
9	45	12	24
10	50	16	27
11	55	5	19
12	60	8	13
13	65	11	10
14	70	12	14
15	75	19	25
16	80	15	11
17	85	22	23
18	90	21	18
19	95	27	16
20	100	57	23

Figure 5: First 100 results from the memory unit, Health is stored in *Value* and wins and losses are stored respectively

The same data from the table is also presented as a visual graph in Figure 6. Notable peaks show that more training is required to create accurate results. Over time the graph will smooth out.

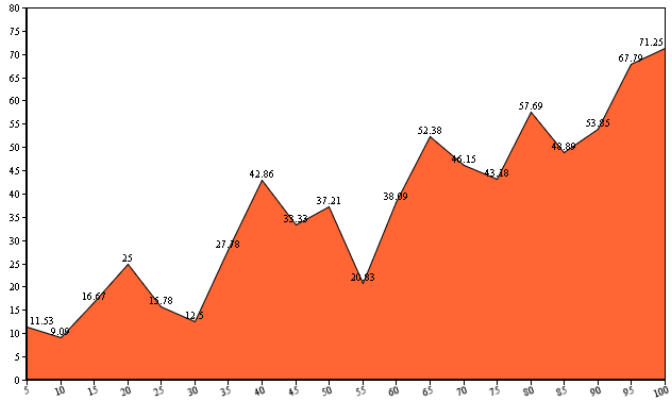

Figure 6: First 100 results from the memory unit as a visual graph. On the vertical is the estimated percentage win rate during an engagement, on the horizontal is the health parameter

8. CONCLUSION

Examining a large variety of possible intelligent agent architectures, BDI has appealed to me the most with its simplicity and yet effectiveness with providing appropriate architecture to replicate human behaviour in First Person Shooter games. Upon experiencing the implementation of the algorithm first hand and the outcome it produced I can conclude in confidence that the overall decision making process of the agent is very rational and ocassionally very unpredictable. The assistance from interviews from human players was invaluable and allowed development of an accurate system. Since every human player has different ways of playing a game and follows a different thought process, the BDI architecture allows to model any particular human behaviour.

However, certain undesirable aspects of the agent that are outside of my control where observed such as: aimbotting, where the AI shoots enemies precisely at long range due to available mathematical accuracy that is not available to human players. Another problem is the articial navigation around the map, due to the default ray tracing module developed by the Pogamut community, the AI moves in the exact center of a coridor in straight lines without jumping, something human players would never do. Since these are the modules that I was not focusing on during development, the AI does not resemble a human player and would possibly fail a turring test. However, I believe the decision making of the AI is very intelligent and rational.

To improve my agent further, individual combat and navigation components could be added. Further machine learning parameters could be included to improve the agents unpredictability. Offline agent training could be implemented to provide accurate results. A BDI architecture language such as GOAL or JACK [Hindriks et al. 2011] [Norling and Sonenberg 2004] could possibly provide a better result, since that is their purpose.

Quite often agent hiccups are noticeable, where the agent has trouble deciding where to go or goes back to the same item. This is due to the fact that the agent makes decisions real time before the decisions go to the execute. A better implementation would be to add an action queue that would be modified based on current situation. This means the agents decisions would be smoother.

APPENDIX

A. REFERENCES

[UT2 2014] 2014. 2kbotprize @ONLINE. (Jan. 2014). http://botprize.org/

[Hun 2014] 2014. Hunter Bot overview @ONLINE. (Jan. 2014). http://pogamut.cuni.cz/pogamut_files/latest/doc/tutorials/ch09s04.html

[Pog 2014] 2014. Pogamut @ONLINE. (Jan. 2014). http://pogamut.cuni.cz/main/tiki-index.php

[Cothran 2014] Jeremy Cothran. 2014. Winning the 2K Bot Prize with a Long-Term Memory Database using SQLite @ONLINE. (Jan. 2014). http://aigamedev.com/open/articles/sqlite-bot/

[Hindriks et al. 2011] KoenV. Hindriks, Birna Riemsdijk, Tristan Behrens, Rien Korstanje, Nick Kraayenbrink, Wouter Pasman, and Lennard Rijk. 2011. Unreal Goal Bots. In *Agents for Games and Simulations II*, Frank Dignum (Ed.). Lecture Notes in Computer Science, Vol. 6525. Springer Berlin Heidelberg, 1–18. DOI: http://dx.doi.org/10.1007/978-3-642-18181-8_1

[Norling and Sonenberg 2004] Emma Norling and Liz Sonenberg. 2004. Creating Interactive Characters with BDI Agents. (2004).

[Rueda et al. 2002] Sonia V. Rueda, Alejandro J. Garcia, and Guillermo R. Simari. 2002. Argument-based Negotiation among BDI Agents. (2002).

[Russell 1997] Stuart Russell. 1997. Rationality and intelligence. *Artificial Intelligence* (1997).

[Sutton and Barto 1998] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press. http://www.cs.ualberta.ca/\%7Esutton/book/ebook/the-book.html

[Umarov and Mozgovoy 2012] Iskander Umarov and Maxim Mozgovoy. 2012. Believable and Effective AI Agents in Virtual Worlds: Current State and Future Perspectives. *IJGCMS* 4, 2 (2012), 37–59. http://dblp.uni-trier.de/db/journals/ijgcms/ijgcms4.html#UmarovM12