Designing BOTs with BDI Agents

Purvag Patel, and Henry Hexmoor
Computer Science Department, Southern Illinois University, Carbondale, IL, 62901, USA
purvag@siu.edu and hexmoor@cs.siu.edu

ABSTRACT

In modern computer games, 'bots' - Intelligent realistic agents play a prominent role in success of a game in market. Typically, bots are modeled using finite-state machine and then programmed via simple conditional statements which are hard-coded in bots logic. Since these bots have become quite predictable to an experienced games player, she might lose her interest in game. We present a model of bots using BDI agents, which will show more human-like behavior, more believable and will provide more realistic feel to the game. These bots will use the inputs from actual game players to specify her Beliefs, Desires, and Intentions while game playing.

KEYWORDS: Agents, BDI, bots, games, collaboration.

1. INTRODUCTION

It is incontrovertible that computer and video game industry have become a billion dollar industry. With the development of 3D game engines, gaming technologies have reached new heights of success. Online gaming community collaborates using contemporary video game consoles, for example, Xbox, Playstations, and Wii. Beyond the gaming aspects these collaborative technologies are sought to be used by The US Army in training staffers in thinking complex military decision making processes. It is time for using these gaming technologies for designing agents for Multi-agents collaboration. We have embarked in formalizing agent based systems that automate the collaborative processes with First Person shooter (FPS) game 'bots'.

A 'bot' (short for robot) is a Non-Player Character (NPC) in a multiplayer video game that is designed to behave similar to a human-controlled player[1]. Examples of such bots are the bots used in games like Counter-Strike[15], Half-Life[16] and Quake[19]. Several bots and their source code are available online to game players and game developers. J. Broome on his website has provided information about development of a bots for the game of Half-Life and had designed several bots for the same[3]. He also explains the creation of 'Half-Life MOD'[1], getting the source code of the game, compiling the code, and finally using it for the bot development or modification. Our main focus for this research is the game called Counter-Strike and its bot, which itself is a MOD for Half-Life and runs on Half-Life game engine. These 3D games provide a readily available real world environment with game maps loaded with artifacts such as boxes, bridges, big gates, and tunnels for hiding, taking covers, and ambush for agents to interact with the world. Therefore, bots can be used to simulate collaborative war theaters wherein we can simulate human-like behavior for coordinating and plan sharing. In addition to this, game developers can use these improved collaborative bots to make games more interesting to game players and increasing their revenue.

In section II, we are providing an overview on the related work carried out by various people. Section II provides a background on the game of counter-strike, which is used as a case study along with a more detailed description on bots and its types. Finally, we characterized the bots as agents and presented a model of BDI agents that can be used to replace bots in section III.

2. RELATEDWORK

N. Cole et. al. argues that to save computation and programmer's time, the game AI uses many hard-coded parameters for bots' logic, which results in usage of enormous amount of time for setting these parameters[4]. Therefore, N. Cole et. al. proposed use of genetic algorithm for the task of tuning these parameters and showed that these methods resulted in bots which are

[1]A Half-Life MOD' is a modification that someone has made to the single player or multiplayer version of the game Half-Life. These MODs usually incorporate new weapons, new levels (maps), and/or new methods or rules for playing the game[3].

competitive with bots tuned by a human with expert knowledge in the game. N. Cole at. al. selected the parameters to tune, allowed them to tune while running genetic algorithms, evolved bot against each other, and finally tested these evolved bots against original bots to test their performance. Another related work is done by S. Zanetti et. al. who used the bot from FPS game Quake 3 and demonstrated the use of Feed Forward Multi-Layer Neural Network trained by a Genetic Algorithm to tune parameters as tuned by N. Cole at. al.[5]. Albeit, their resulting bot did not reach the competitive playable level.

Nareyek argues that role of AI techniques currently used in game AI is very different from those studied in the academic AI [6]. Nareyek used Wooldridge and Jenning's work as starting point and classified game agents according to their trade-offs between computation time and the realization of sophisticated goal-oriented behavior[8]. Narayek classified agents as reactive agents, triggering agents, deliberative agents, hybrid agents, and anytime agents [6]. Nareyek also provided insight on how autonomous agents (bots in our case) can be used in games[6]. Nareyek's work guides us to a new direction of programming for games agents.

BDI model of human rational actions was originally developed by M. Bratman which has been adopted by agent community[7]. M. Wooldridge says following about an agent[9]:

> *"Intuitively, an agent's 'beliefs' correspond to information the agent has about the world. These beliefs may be incomplete or incorrect. An agent's 'desires' represents states of affairs that the agent would, in an ideal world, wish to be brought about. Finally, an agent's 'intentions' represent desires that it has committed to achieving."*

There are efforts to use the BDI model of rational agency to implement a bot in FPS games.

A. Bartish et. al. in his experiment showed how the choice of implementation, i.e. agents and FSM, may affect the performance and complexity of games. They proved that complexity measure of function of various numbers of behaviors was linear for agents and quadratic for FSM. Although runtime performance is comparable for small number of entities, it degrades at higher rate for agents[10].

N.P. Davies et. al. researched creating a human-like AI, based on BDI paradigm, and design of framework for implementing deliberative agents in computer games[11][12]. To test the behavior of AI they proposed an architecture using JACK[14] and linked it to Unreal Tournament[17] game engine via use of GameBots/JavaBot[18] technology which they used in both of their works [11][12]. E. Norling et. al. in a similar work presents the usage of BDI agents for development of human-like synthetic characters using JACK[13]. They modeled expert players in existing game to build bots in game of Quake 2 and demonstrated that same techniques can be used to build complete original characters in games. E. Norling et. al. used a form of knowledge elicitation known as 'Applied Cognitive Task Analysis' (ACTA) to capture player's strategies[13]. In interview with the players, with different playing styles, they presented them sets of questions to know their reaction in a particular situation[13]. Based on that they demonstrated how the BDI paradigm facilitated the capture of the strategic thinking of the players using JACK programming language for implementation.

3. BACKGROUND

3.1. Game of Counter-Strike

It is necessary to study the environments in which bot need to interact with and know how they are used in computer games. Therefore, we selected the game of Counter-Strike as a case study for our project which is one of most popular and open source computer game played by thousands of players simultaneously on internet. Basically Counter-Strike is a 'team based' first-person shooter game with two teams Terrorist and Counter-Terrorist. Each team consists of five players, but depending on game play team size might be set up to 30 players per team. Both the teams play against one another and the team that wins more number of rounds is the winner. Normally, teams play approximately ten to fifteen rounds within an hour.

Figure 1 shows a standard map of Counter-Strike called DE_DUST. On the map, two sites labeled A, and B are bomb sites where a terrorist makes effort to plant bomb. On the contrary, a counter-terrorist makes effort to defend these bomb sites and if the bomb gets planted by terrorist counter-terrorist tries to defuse the bomb before it explodes. In the beginning of each round, both the teams are located at designated locations on map, for example, again the position marked by label A in figure 1 is 'counter-terrorist camp' from where the counter-terrorist starts the round. Once the round starts they start moving around the map, fighting with each other and try to achieve their respective goals. Similarly, the position marked by label B on in figure 1 is 'terrorist camp' for terrorists. Each map in the game has many paths/plans that players may use to go from their base camp to target place and usually they use different path/plans for each round. Again, figure 1 shows two such 'critical position', marked by arrows label W, and X, on terrorist path

through which terrorist needs to pass through to reach the bomb sites and counter-terrorist having the knowledge of these positions would plan to defend at these places. There are many other such 'critical positions' on the map, for example bomb sites.

Figure 1. Game Map

3.2. Bots in Computer Games

Bots in counter-strike, also called NPCs are used to replace human players. Bots play as a part of the team and achieve goals similar to humans. Bots simulate human players and are aimed to give game players 'illusion' of playing against actual human player similar to computer in the Turing test. Currently, bots used in counter-strike are programmed to find the path, attack opponent players, or run away from the site if they have heavy retaliation or if their energy is less providing an illusion they are intelligent. Similar species of bots are also used in many other FPS games, with similar method of programming.

Bots are usually pre-programmed according to the requirements of a game and play for or against human players. Based on how bots are programmed, there can be two styles of bots[2]:

1. Static: Static bots are static in the levels and maps have already been processed. This means that they need to have all information about the maps and level prior to the start of game.
2. Dynamic: Dynamic bots learn as they go through the level. They can be played at any level while static bots cannot.

Both these techniques produce good quality bots, with a single difference that dynamic bots can learn through level while static cannot.

Usually, bots in computer games are modeled using a FSM as shown in figure 2 where rectangle represents a possible state whereas leading edges shows transition between states. It is just a miniature representation of actual bot where many more such states exist with more complicated transitions. FSM for bots is quite self explanatory wherein first the bot start by making initial decisions viz. game strategies, buying weapon, etc. and then start searching for enemies. Once the enemy is spotted it makes a transition to attack state in which she fires bullets at enemy. A Bot may kill an enemy; therefore in that case it will again start searching for enemy as shown in figure 2. Also, a bot could be in any of the above mentioned states and might get killed by the enemy.

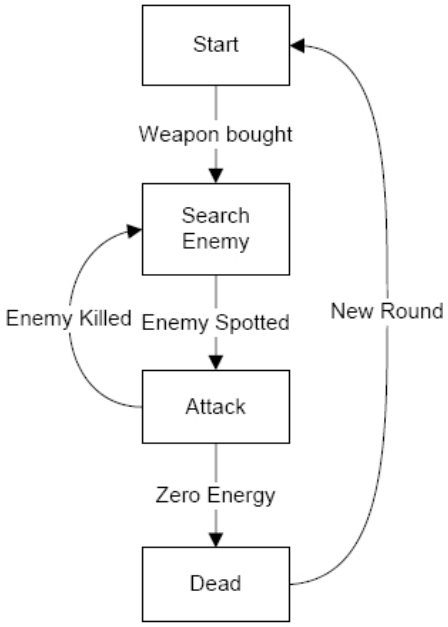

Figure 2. A prototypical FSM for a bots

FSM for bots are implemented using simple if-else or switch-case statements usually using c/c++ programming language. Problem with this style of programming is that their behavior becomes very predictable to even little experienced players. Players will be able to predict their action and what path they will use. In addition to this, the main problem is that all the team members will act in a similar manner and will follow same strategy. This is not in the case with human players where all players play differently irrespective of playing styles of other teammates. Some players may play aggressively while others may play cautiously and it also depends on the type of weapon a player may select. Bots need to be made more sophisticated to improve gaming experience and keep players' interest alive in the game by making them more realistic.

4. APPROACH

4.1. Bots as Agents

View of a bot as a typical agent which operates in a world or a game map, scan percept from environment, and is able to take autonomous decision viz. whether to attack, hide, or reload gun is show in figure 3. Objectives of the agent in games similar to Counter-strike is to

- kill maximum number of enemies, and
- accomplish their goals, i.e. to plant the bomb or to defuse the bomb.

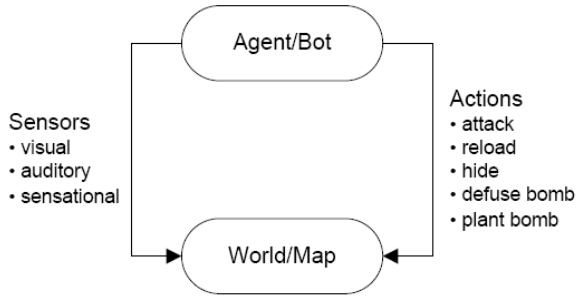

Figure 3. Bot actions and sensors

Agent needs to be programmed to 'sense or observe' the world through its visual, auditory, and sensational sensors. Agent will get the 'percept' about the world through these 'sensors', for example, an agent may spot an enemy or friend, hear a gun fire, sense that she herself was hit, etc.. Based on her 'percepts', agent will make decisions and perform 'actions' such as whether to attack, hide, reload, and defuse the bomb or plant the bomb. Many agents will be simultaneously active in the world and their 'actions' will affect world. All bots will have no control on 'action' of other agent in the world making each agent autonomous. Even though, a game world is predictable to an agent as she has knowledge of effects of her action on the world, still it is unpredictable as she cannot predict the actions of other agents.

We claim that this agent which will be used to replace bot will have following four classic agent properties:

1. autonomy: an agent will be autonomous as she needs to make her own decisions based on her own beliefs and goals. There is no other authority that may affect her decision making process.

2. proactiveness: an agent has a very 'definite goals', for example, to win the round, which makes him a proactive agent.

3. reactiveness: an agent will have to abort her current plan if she deliberates that some other plan is more fruitful. For example, if bot is currently searching for enemy and she finds out that bomb is planted then she will have to abort her current plan and start running toward bomb sites to defuse.

4. social ability: an agent certainly has social ability, she needs to communicate with other team members via audio messages on radio or by sending typed messages.

Having viewed bot as agent, in next section we will see more specifically, the model of agent that can be used to replace currently bot.

4.2. Model of BDI agent for bot

Humans have ability to perform multiple task simultaneously. A critical observation may lead humans to abort their current task to re-think and start a different task. For example, a player while defusing a bomb may notice that opposite team player is about to attack her, and she may stop defusing the bomb to attack back at opponent. This basic observation leads us to design a BDI agent which can perform multiple tasks simultaneously for which an agent will fork two distinct processes which will run simultaneously.

- First process will perform the task of observing the world, and
- second process will be responsible for agent's deliberation, and means-ends reasoning.

We propose to use Algorithm 1 and 2, each corresponding to one process respectively which are adopted from M. Wooldridge's Agent Loop Algorithm for the agents. Agent will fork two processes, first process viz. algorithm 2 for observing the world, and updating the internal beliefs while second viz. algorithm 1 will perform the tasks of deliberating and means-end reasoning. Both these processes will share following data structure:

- agent's Belief set(B),
- agent's Intention set(I), and
- semaphore(*reconsiderflag*) an agent will use make reconsideration decision.

In algorithm 1, agent runs a continuous loop starting with initial beliefs(B_o), and initial intentions(I_o). Agent deliberates about her desires, generates her intentions based on those desires, selects a plan to achieve those intentions, and finally execute that plan. In algorithm 1,

agent does not observe the world or updates her beliefs which are done in algorithm 2 that we will discuss in due course.

Algorithm 1 Agent Control Loop

```
1: B := B_o /*B_o are initial beliefs */
2: I := I_o /*I_o are initial beliefs */
3: while true do
4:    D := options(B, I)
5:    I := filter(B, D, I)
6:    π := plan(B, I)
7:    while not empty(π) or succeeded(I, B) or
      impossible(I, B) do
8:       α := hd(π)
9:       execute(α)
10:      if reconsiderflag then
11:         D := options(B, I)
12:         I := filter(B, D, I)
13:         reconsiderflag := false
14:      end if
15:      if not sound(π, I, B) then
16:         π = plan(B, I)
17:      end if
18:      π := tail(π)
19:   end while
20: end while
```

Agent deliberates in two steps, 'option generation', and 'filtering'. In 'option generation' agents generated set of possible alternatives using function '*options(...)'* which produces desires of agent. Function option will have signature

$$options : \gamma(Bel) \times \gamma(Int) \rightarrow \gamma(Des).$$

Next step is 'filtering' in which agent chooses between current desires and commit to achieving few of them producing a set of intentions using function '*filter(...)*' with signature

$$filter : \gamma(Bel) \times \gamma(Des) \times \gamma(Int) \rightarrow \gamma(Int).$$

After deliberation, next step for agent is means-ends analysis which results in selection of a plan(*π*) - a set of actions, to accomplish those generated intentions. After plan is selected, the agent picks one action from the head(*hd(π)*) of action set in plan and executes that action. Agent will continue this process until she"

- has executed all actions in plan (*empty(π)*),
- has achieved current intentions (*succeeded(I,B)*), and
- believes current intentions are not possible (*impossible(I,B)*).

In each step agent checks semaphore - *reconsiderflag*, if the flag is set, then agent stops to re-deliberate her desires and intentions. Deliberation is a costly process, hence, an agent deliberates only when it is absolutely necessary i.e. when shared semaphore - *reconsiderflag* is set, which in turn is modified by a loop in algorithm 2. Finally, agent checks whether her current plan is sound(*sound(π,I,B)*) with her current intentions and beliefs, and if not she re-plans i.e. performs means-end analysis.

Now, agent's second process runs another continuous loop which is shown in algorithm 2. This loop is faster than the loop in algorithm 1, and performs the task of belief update and reconsideration. At each step in this algorithm agent observe the world to get the next percept, and based on that percept agent updates her beliefs. Now an agent uses Boolean function '*reconsider(..)*', which returns true if agent needs to reconsider its intention based on her current beliefs, and intention. In that case, agent will set shared semaphore i.e. 'reconsiderflag' to true and fire a trigger which will force agent to abort its current action. Again, function '*execute(...)*' used in algorithm 1 is not an atomic function which will be programmed to abort after the ABORT trigger is fired by second process.

Algorithm 2 Agent Analyze Percept Loop

```
1: while true do
2:    get next percept ρ
3:    B := brf(B, ρ)
4:    if reconsider(I, B) then
5:       reconsiderflag := true
6:       trigger ABORT current action
7:    end if
8: end while
```

Notice that, agent sets the 'reconsiderflag' to true before triggering the ABORT command. Hence, next step an agent will perform in process one is of re-deliberation i.e. generating new desires and intentions. This is absolutely necessary because an agent now believes that an important event occurred in the world and agent needs to re-deliberate.

Original M. Wooldridge's algorithm would not stop to scan next percept and re-consider until agent completely executes her current action. Separation of the two processes is necessary for our agents because agent should not continue executing her current action if she believes that there is other action with more utilities is possible. This behavior is desired for our agent as the world is very dynamic and agent cannot predict the behavior of other agents playing in map. Again, the world being very dynamic it is also not necessary that with every new percept agent needs to stop for reconsideration. Let us consider an example to support this approach. Suppose counter-terrorist agent is executing her current action which is 'going to terrorist's camp' and she spots a

terrorist agent on the way. An ideal behavior will be to stop and start attacking terrorist agent which would be achieved with current approach. While with original algorithm, agent would continue executing her current action i.e. 'going to terrorist camp' and on reaching the camp she would re-consider and find out that she spotted terrorist agent on the way which will be late to respond. In a similar case, if agent spots friendly agents, then she need not stop to reconsider; she can continue doing her current action i.e. of going to terrorist's camp.

4.3. Usage of BDI model for bots

Basically, we want our bots to behave similar to actual game players who are experts and play the game in teams. At the start of every round, players collaborate and deliberate about team strategy; for example, whether they would play aggressive or defensive. Apart from the collaborative team strategy each player has her own style of playing, again a player may be aggressive or defensive which also may depend on the type of weapon they buy at the start of each round. With our BDI model, we want to capture this basic difference in player's strategies. Each agent in the team having different style of playing based on the inputs from experts will collaborate with one another to achieve team's goal.

All the bots regardless of what collaborative strategies they may employ, they will have similar sets of beliefs and intentions. For example, at any time period a counter-terrorist agent may have beliefs and intentions as shown in figure 4. Basically, a game programmer need to build a generic module for scanning percept and belief update, which will remain same irrespective of the player's strategies, i.e. process two of our algorithm is reusable for all bots. Even the '*reconsider(...)*' function need not to be changed because an 'urgent situation' for one agent will also be an 'urgent situation' for another agent no matter what strategy agent may use.

For two distinct agents to capture different styles, viz. aggressive or cautious, both of them will have to produce different desires. Thus, the only function which needs to be customized to produce bots with different collaborative style of playing will be function '*options(...)*'. This function will generate different sets of desires accounting for type of bot we want to model as shown in figure 5. Notice that a more aggressive agent will have desire to kill more players, on the contrary a cautious agent would like to go to specific spots and wait for her target. In the future, agent is supposed to produce intentions which are subset of her desires. In this manner we will be able to capture different collaborative styles of playing of different players.

$Bel\ Loc(Terrorist(X), Area(A))$ $Bel\ Loc(Terrorist(Y), Area(B))$ $Bel\ Loc(CounterTerrorist(X), Area(B))$ $Bel\ Group - Strategy(\Delta))$
(a) Beliefs
$Int\ Kill(Terrorist(X))$ $Int\ Kill(Terrorist(Y))$ $Int\ Defend(BombSite(A))$
(b) Intentions

Figure 4. Bots' Belief and Intention Set

$Des\ Kill(Terrorist(X))$ $Des\ Kill(Terrorist(Y))$
(a) Aggressive
$Desire\ Defend(BombSite(A))$ $Desire\ Defend(BombSite(B))$
(b) Cautious

Figure 5. Bots' desire set

We are planning on implementing above mentioned approach using JAVA. We will develop a miniature simulation of Counter-Strike using JAVA applets to simulate agents and its environment. Even though, our simulation will be small as compared to real game Counter-Strike, the usage of Algorithm 1 and 2 to prove its effectiveness can be demonstrated. The reason is there already are related work ([10],[11],[12], and [13]) that have demonstrated the integration of JAVA and Game Engines to implement BDI agent for bots in games, to proves that our miniature simulation of the game can be easily extended for use with current Game Engines in future. Henceforth, with this research we demonstrate the use of enhanced BDI model to fit the needs of bots in games and make them more human-like, believable, and to provide more realistic feel to the game.

5. CONCLUSION

There is a strong affinity between online bots and agents. We have proposed a BDI agent system that models computational bots. Current bots system is very predictable. After a player spends couple of days playing the game she can predict the behavior of all the bots. Furthermore, all the bots will behave in a similar fashion. This makes the game less interesting to the game players who eventually may lose interest in the game. Our BDI model will make bots more human-like, believable, and will provide more realistic feel to the game. BDI agents are computationally inefficient, but with growing processing power game developers can afford to dedicate

additional CPU cycles to these agents. We have not yet explored the machine learning techniques, like neural network or genetic algorithm, which can make bots adapt the game playing strategies against human's players.

REFERENCES

[1] Valve Developer Community, viewed 5th oct. 2008. http://developer.valvesoftware.com/wiki/Bots

[2] THE BOT FAQ, viewed 5th oct. 2008. http://members.cox.net/randar/botfaq.html

[3] J. Broome. Botman's bots Half-Life bot development, http://botman.planethalflife.gamespy.com/index.shtml

[4] N. Cole, S. J. Louis, and C. Miles, "Using a Genetic Algorithm to Tune First-Person Shooter Bot," In Proceedings of the International Congress on Evolutionary Computation, 2004

[5] S. Zanetti and A. Rhalibi, "Machine Learning Techniques for FPS in Q3," *ACE'04*, June 3-5. 2004, Singapore

[6] A. Nareyek, "Intelligent Agents for Computer Games," In *Marsland, T. A., and Frank, I. (eds.), Computers and Games, Second International Conference*, CG 2000, Springer LNCS 2063, 414-422.

[7] M. Bratman, INTENTIONS, PLANS, AND PRACTICAL REASON, CLSI Publications, 1999

[8] M. Wooldridge, and N. R. Jenning, "Intelligent Agents: Theory and practice," Knowledge Engineering Review Volume 10 No 2, June 1995.

[9] M. Wooldridge, REASONING ABOUT RATIONAL AGENTS, Massachusetts Institute of Technology, 2000, pages 1-45

[10] A. Bartish, and C. Thevathayan, "BDI Agents for Game Development," *AAMAS'02*, July 15-19. 2002, Bologna, Italy

[11] N.P. Davies, Q.H. Mehdi, and N. Gough, "Creating and Visualizing an Intelligent NPC using Game Engine and AI Tool," *European Conference on Modeling and Simulation*, ECMS, 2005, Riga, Latvia, 721-726, ISBN 1-84233-112-4

[12] N.P. Davies, Q.H. Mehdi, and N. Gough, "A Framework for Implementing Deliberative Agents in Computer Games," In Proceedings *20th European Conference on Modeling and Simulation Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel l' ECMS*, 2006 ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

[13] E. Norling, and L. Sononberg, "Creating Interactive Characters with BDI Agents," In Proceedings of the *Australian Workshop on Interactive Entertainment IE*, 2004

[14] Agent Oriented Software Group, JACK Intelligent Agents(2006).

[15] Valve Corporation. Counter-Strike: Source. www.counter-strike.net/

[16] Valve Corporation. Half-Life II. http://orange.half-life2.com/

[17] Midway Home Entertainment, Inc. Unreal Tournament 3. http://www.unrealtournament3.com/

[18] Gamebots. http://gamebots.planetunreal.gamespy.com/index.html

[19] Id Software, Inc. Quake II. http://www.idsoftware.com/games/quake/quake2/