

# Goal-Driven Autonomy with Case-Based Reasoning

Héctor Muñoz-Avila<sup>1</sup>, Ulit Jaidee<sup>1</sup>, David W. Aha<sup>2</sup>, and Elizabeth Carter<sup>1</sup>

<sup>1</sup>Department of Computer Science & Engineering;  
Lehigh University; Bethlehem, PA 18015

<sup>2</sup>Navy Center for Applied Research in Artificial Intelligence;  
Naval Research Laboratory (Code 5514); Washington, DC 20375  
ulj208@lehigh.edu | munoz@cse.lehigh.edu | david.aha@nrl.navy.mil | eec209@lehigh.edu

**Abstract.** The vast majority of research on AI planning has focused on automated plan recognition, in which a planning agent is provided with a set of inputs that include an initial goal (or set of goals). In this context, the goal is presumed to be static; it never changes, and the agent is not provided with the ability to reason about whether it should change this goal. For some tasks in complex environments, this constraint is problematic; the agent will not be able to respond to opportunities or plan execution failures that would benefit from focusing on a different goal. *Goal driven autonomy* (GDA) is a reasoning framework that was recently introduced to address this limitation; GDA systems perform anytime reasoning about what goal(s) should be satisfied [4]. Although promising, there are natural roles that case-based reasoning (CBR) can serve in this framework, but no such demonstration exists. In this paper, we describe the GDA framework and describe an algorithm that uses CBR to support it. We also describe an empirical study with a multiagent gaming environment in which this CBR algorithm outperformed a rule-based variant of GDA as well as a non-GDA agent that is limited to dynamic replanning.

## 1 Introduction

One of the most frequently cited quotes from Helmuth von Moltke, one of the greatest military strategists in history, is that “no plan survives contact with the enemy” [1]. That is, even the best laid plans need to be modified when executed because of (1) the non-determinism in one’s own actions (i.e., actions might not have the intended outcome), (2) the intrinsic characteristics of adversarial environments (i.e., the opponent might execute unforeseen actions, or even one action among many possible choices), and (3) imperfect information about the world state (i.e., opponents might be only partially aware of what the other side is doing).

As a result, researchers have taken interest in planning that goes beyond the classic deliberative model. Under this model, the state of the world changes solely as a result of the agent executing its plan. So in a travel domain, for example, a plan may include an action to fill a car with enough gasoline to follow segments (A,B) and (B,C) to drive to location C from location A. The problem is that the dynamics of the environment might change (e.g., segment (B,C) might become unavailable due to some road damage). Several techniques have been investigated that respond to contingencies which may invalidate the current plan during execution. This includes *contingency planning* [2], in which the agent plans in advance for plausible contingencies. In the travel example, the plan might include an alternative subplan should (B,C) becomes unavailable. One such subplan might call to fill up with more gasoline at location B and continue using the alternative, longer route (B,D), (D,C). A

drawback of this approach is that the number of alternative plans required might grow exponentially with the number of contingencies that need to be considered. Another alternative suggested is *conformant planning* [3], where the generated plan is guaranteed to succeed. For example, the plan might fill up with enough gasoline at B so that, even if it has to go back to B after attempting to cover the segment (B,C), it can continue with the alternative route (B,D), (D,C). The drawback is that the plan might execute many unnecessary steps for contingencies that do not occur (such as obtaining additional gasoline while initially in location B).

Recently, another alternative, called *Goal Driven Autonomy* (GDA), was proposed to solve this problem [4, 5]. GDA agents continuously monitor the current plan's execution and assess whether the actual states visited during the current plan's execution match expectations. When mismatches occur (i.e., when the state does not meet expectations), a GDA monitor will suggest alternative goals that, if accomplished, would fulfill its overarching objectives. In our travel example, the GDA monitor might suggest that the agent first drive to location D and then to C.

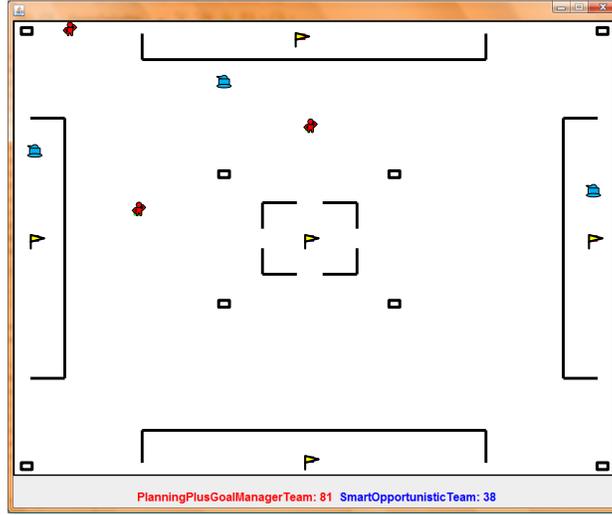
In this paper we introduce and assess CB-gda, the first GDA system to employ case-based reasoning (CBR) methods [6]. CB-gda uses two case bases to dynamically generate goals. The first case base relates goals with expectations, while the latter's cases relate mismatches with (new) goals. We describe an empirical study of CB-gda on the task of winning games defined using a complex gaming environment (DOM). Our study revealed that, for this task, CB-gda outperforms a rule-based variant of GDA when executed against a variety of opponents. CB-gda also outperforms a non-GDA replanning agent against the most difficult of these opponents and performs similarly against the easier ones. In direct matches, CB-gda defeats both the rule-based GDA system and the non-GDA replanner.

The rest of the paper continues as follows. Section 2 describes a testbed that we use for our experiments and for motivating some of the GDA concepts. Section 3 presents the GDA framework. In Section 4, we discuss related work. In Section 5, we introduce our case-based GDA algorithm and give an example of its behavior in Section 6. Finally, Section 7 presents our empirical study and Section 8 discusses the results and plans for future work.

## 2 DOM Environment

Domination (DOM) games are played in a turn-based environment in which two teams (of *bots*) compete to control specific locations called *domination points*. Each time a bot on team  $t$  passes over a domination point, that point will belong to  $t$ . Team  $t$  receives one point for every 5 seconds that it owns a domination point. Teams compete to be the first to earn a predefined number of points. Domination games have been used in a variety of combat games, including first-person shooters such as Unreal Tournament and online role-playing games such as World of Warcraft.

Domination games are popular because they reward team effort rather than individual performance. No awards are given for killing an opponent team's bot,

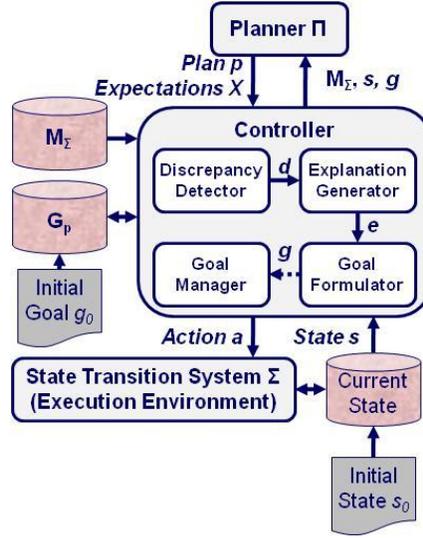


**Figure 1:** An example DOM game map with five domination points (yellow flags), where small rectangles identify the agents' respawning locations, and the remaining two types of icons denote each player's agents.

which *respawns* immediately in a location selected randomly from a set of map locations, and then continues to play. Killing such bots might be beneficial in some circumstances, such as killing a bot before she can capture a location, but the most important factor influencing the outcome of the game is the strategy employed. An example strategy is to control half plus one of the domination locations. A location is captured for a team whenever a bot in that team moves on top of the location and within the next 5 game ticks no bot from another team moves on top of that location. Figure 1 displays an example of DOM map with five domination points [7].

Bots begin the game and respawn with 10 health points. Enemy encounters (between bots on opposing teams) are handled by a simulated combat consisting of successive die rolls, each of which makes the bots lose some number of health points. The die roll is modified so that the odds of reducing the opponent health points increase with the number of friendly bots in the vicinity. Combat finishes when the first bot health points decreases to 0 (i.e., the bot dies). Once combat is over, the death bot is respawned from a spawn point owned by its team in the next game tick. Spawn point ownership is directly related to domination point ownership, if a team owns a given domination point the surrounding spawn points also belong to that team.

The number of possible states in DOM is a function of (1) the number of cells in the map, which is  $n * m$  where  $n$  is the number of rows and  $m$  is the number of columns, (2) the number,  $b$ , of bots in each team, (3) for each bot the number of health points (between 0 and 10), (4) the number,  $t$ , of teams, (5) the number,  $d$ , of domination locations, (6) a number between 0 and 5 for each domination location; 0 indicates that no other team is trying to capture the location and 1 to 5 indicates the number of game ticks since a bot has attempted to capture the location belonging to a different team. The total number of states is  $(n \times m)^{(b \times t)} \times 11^{(b \times t)} \times 6^d \times (t+1)^d$ . The exponent  $(t+1)$  accounts for the beginning of the game in which the domination locations do not belong to any team. In our experiments,  $n = m = 70$ ,  $b = 3$ ,  $t = 2$ , and  $d = 4$ . Hence, the number of possible states is  $2 \times 10^{34}$ .



**Figure 2:** The GDA conceptual model

Because of the large number of possible states, we follow the state abstraction model of Auslander *et al.* (2008) for decision making by the AI agents in DOM games [7]. Since there is no reward for killing an opponent, emphasis is made into controlling the domination locations. Hence the states in the abstract model simply indicate which team owns the domination locations, reducing the number of states to  $d^{(t+1)}$ . The decisions that the agent makes is to decide to which domination location to send each bot. Thus, in the abstract model the number of actions is  $(b \times t)^d$ .

DOM is a good testbed for testing algorithms that integrate planning and execution because domination actions are *non-deterministic*; if a bot is told to go to a domination location the outcome is uncertain because the bot may be killed along the way. Domination games are also *adversarial*; two or more teams compete to control the domination points. Finally, domination games are *imperfect information* games; a team only knows the locations of those opponent bots that are within the range of view of one of the team's own bots.

### 3 Goal Driven Autonomy

Goal driven autonomy permits autonomous agents to direct the focus of their planning activities, and thus become more self-sufficient.

**Definition:** *Goal Driven Autonomy* (GDA) is a goal reasoning method for problem solving in which autonomous agents dynamically identify and self-select their goals throughout plan execution.

The GDA conceptual model has four steps, as shown within the Controller component in Figure 2. This model extends the conceptual model of online classical planning [8], whose components include a Planner, a Controller, and a State Transition System  $\Sigma = (S, A, E, \gamma)$  with states  $S$ , actions  $A$ , exogenous events  $E$ , and state transition function  $\gamma: S \times (A \cup E) \rightarrow 2^S$ . In the GDA model, the Controller is centric. It receives as input a planning problem  $(M_\Sigma, s_c, g_c)$ , where  $M_\Sigma$  is a model of  $\Sigma$ ,  $s_c$  is the

current state (e.g., initially  $s_c$ ), and  $g_c \in G$  is a goal that can be satisfied by some set of states  $S_g \subseteq S$ . It passes this problem to the Planner  $\Pi$ , which generates a sequence of actions  $A_c = [a_{c_1}, \dots, a_{c+n}]$  and a corresponding sequence of expectations  $X_c = [x_{c_1}, \dots, x_{c+n}]$ , where each  $x_i \in X_c$  is a set of constraints that are predicted to hold in the corresponding sequence of states  $[s_{c+1}, \dots, s_{c+n+1}]$  when executing  $A_c$  in  $s_c$  using  $M_\Sigma$ . The Controller sends  $a_c$  to  $\Sigma$  for execution and retrieves the resulting state  $s_{c+1}$ , at which time it performs the following knowledge-intensive (and GDA-specific) tasks:

1. *Discrepancy detection*: GDA detects unexpected events before deciding how to respond to them. This task compares observations  $s_{c+1}$  with expectation  $x_c$ . If one or more discrepancies (i.e., unexpected observations)  $d \in D$  are found in  $s_{c+1}$ , then explanation generation is performed to explain them.
2. *Explanation generation*: This module explains a detected discrepancy  $d$ . Given also state  $s_c$ , this task hypothesizes one or more explanations  $ex \in Ex$  of their cause.
3. *Goal formulation*: Resolving a discrepancy may warrant a change in the current goal(s). This task generates goal  $g \in G$  given a discrepancy  $d$ , its explanation  $ex$ , and current state  $s_c$ .
4. *Goal management*: New goals are added to the set of pending goals  $G_p \subseteq G$ , which may also warrant other edits (e.g., removal of other goals). The Goal Manager will select the next goal  $g' \in G_p$  to be given to the Planner. (It is possible that  $g = g'$ .)

The GDA model makes no commitments to the choice of Planner or algorithms for these four tasks. For example, Muñoz-Avila *et al.* (2010) describe GDA-HTNbots [4], a system that implements a simple GDA strategy in which  $\Pi = \text{SHOP}$  [9], the Discrepancy Detector triggers on any mismatch between the expected and current states, a rule-based reasoner is used for the Explanation Generator and Goal Formulator, and the Goal Manager simply replaces the current goal with any newly formulated goal. However, CBR is not employed in GDA-HTNbots.

## 4 Related Work

Cox's (2007) investigation of self-aware agents inspired the conception of GDA [10], with its focus on integrated planning, execution, and goal reasoning. Some of the terms we adopt, such as expected and actual states, are directly borrowed from that work.

In the introduction we discussed two alternatives to GDA: contingency planning and conformant planning. Their main drawback is that, before plan execution, they require the a priori identification of possible contingencies. In DOM games, a plan would need to determine which domination points to control, which locations to send a team's bots, and identify alternative locations when this is not possible. An alternative to generating contingencies beforehand is performing plan repair. In plan repair, if a mismatch occurs during plan execution (i.e., between the conditions expected to be true to execute the next action and the actual world state), then the system must adapt the remaining actions to be executed in response to the changing circumstances [11, 12]. The difference between plan repair and GDA is that plan repair agents retain their goals while GDA agents can reason about which goals should be satisfied. This also differentiates GDA from replanning agents, which execute a plan until an action becomes inapplicable. At this point, the replanning agent simply generates a new plan from the current state to achieve its goals [13, 14, 15].



The algorithm above displays our CBR algorithm for GDA, called *CB-gda*. It runs the game  $D$  for the GDA-controlled agent  $A$ , which is ordered to pursue a goal  $g_{init}$ . Our current implementation of  $A$  is a case-based planner that searches in the case base PCB for a plan that achieves  $g_{init}$ . The call to  $\text{run}(D, A, g_{init})$  represents running this plan in the game. (Line 1). While the game  $D$  is running (Line 2), the following steps are performed. Variables  $s_i$  and  $g_i$  are initialized with the current game state and agent's goal (Line 3). The inner loop continues running while  $A$  is attempting to achieve  $g_i$  (Line 4). The algorithm waits a time  $t$  to let the actions be executed (Line 5). Given the current goal  $g_i$  and the current state  $s_i$ , agent  $A$  searches for a case  $(s_c, g_c, e_c, pl)$  in PCB such that the binary relations  $\text{SIM}_s(s_i, s_c)$  and  $\text{SIM}_g(g_i, g_c)$  hold and returns the expected state  $e_c$ .  $\text{SIM}(a, b)$  is currently an equivalence relation. (Line 6). We follow the usual textbook conventions [20] to define  $\text{SIM}(a, b)$ , which is a Boolean relation that holds true whenever the parameters  $a$  and  $b$  are similar to one another according to a similarity metric  $\text{sim}(\cdot)$  and a threshold  $th$  (i.e.,  $\text{sim}(a, b) \geq th$ ). Since the similarity function is an equivalence relation, the threshold is 1. The current state  $s_D$  in  $D$  is then observed (Line 7). If the expectation  $e_c$  and  $s_D$  do not match (Line 8), then a case  $(m_c, g_c)$  in MCB is retrieved such that mismatch  $m_c$  and  $\text{mismatch}(e_c, s_D)$ , are similar according to  $\text{SIM}_m(\cdot)$ ; this returns a new goal  $g_c'$  (Line 9). Finally,  $D$  is run for agent  $A$  with this new goal  $g_c'$  (Line 10). The game score is returned as a result (Line 11).

From a complexity standpoint, each iteration of the inner loop is dominated by the steps for retrieving a case from PCB (Line 6) and from MCB (Line 9). Retrieving a case from PCB is of the order of  $O(|\text{PCB}|)$ , assuming that computing  $\text{SIM}_s(\cdot)$  and  $\text{SIM}_g(\cdot)$  are constant. Retrieving a case from MCB is of the order of  $O(|\text{MCB}|)$ , assuming that computing  $\text{SIM}_m(\cdot)$  is constant. The number of iterations of the outer loop is  $O(N/t)$ , assuming a game length of time  $N$ . Thus, the complexity of the algorithm is  $O((N/t) \times \max\{|\text{PCB}|, |\text{MCB}|\})$ .

We claim that, given sufficient cases in PCB and MCB, *CB-gda* will successfully guide agent  $A$  in accomplishing its objective while playing the DOM game. To assess this, we will use two other systems for benchmarking purposes. The first is HTNbots [13], which has been demonstrated to successfully play DOM games. It uses Hierarchical Task Network (HTN) planning techniques to rapidly generate a plan, which is executed until the game conditions change, at which point HTNbots is called again to generate a new plan. This permits HTNbots to react to changing conditions within the game. Hence, it is a good benchmark for *CB-gda*. The second benchmarking system is GDA-HTNbots [4], which implements a GDA variant of HTNbots using a rule-based approach (i.e., rules are used for goal generation), in contrast to the CBR approach we propose in this paper.

## 6 Example

We present an example of *CB-gda* running on the DOM game. Suppose there are 3 domination points in the current instance of the game:  $dom_1$ ,  $dom_2$ , and  $dom_3$ . As we explained before, the possible states that are modeled by the case-based agent is the Cartesian product  $\prod_i o_i$  of the owner  $o_i$  of the domination point  $i$ . For instance, if there are 3 domination points, the state  $(E, F, F)$  denotes the state where the first domination point is owned by the enemy and the other two domination points are owned by our friendly team. Suppose that the case base agent was invoked with the goal  $g_{init} =$

*control-dom<sub>1</sub>*, which sets as its goal to control *dom<sub>1</sub>*. Suppose that this is the beginning of the game, so the starting state is  $(N,N,N)$ , indicating that no team controls any of the domination points. Suppose that the case-based agent A retrieves a case  $(s_c, g_c, e_c, pl)$  in PCB such that  $g_c = control-dom_1$  and  $s_c = (N,N,N)$ . Thus,  $pl$  is executed in Line 1 and  $s_i = (N,N,N)$  and  $g_i = control-dom_1$  in Line 3.

After waiting for some time  $t$ , the PCB case base is consulted for a similar case (Line 6). Assume we retrieve the same case as before:  $(s_c, g_c, e_c, pl)$ , where  $s_c = s_i$ ,  $g_c = g_i$ , and  $e_c = (F,N,N)$ . This case says that with this state and with this goal, the **expected** state is one where the controlled team owns *dom<sub>1</sub>*. Now suppose that the **current** state  $s_D$  as obtained in Line 7 is  $(E,N,N)$ , which means that  $s_D$  differs from  $s_c$  (Line 8). At this point, a case is searched in the MCB case base (Line 9). Suppose that a case  $(m_c, g_c)$  exists (and is retrieved) such that  $m_c = (F/E,-,-)$ , which means there is only a mismatch in the ownership of *dom<sub>1</sub>*. Suppose that  $g_c = control-dom_2-and-dom_3$ , a goal that tells the case-based agent to control *dom<sub>2</sub>* and *dom<sub>3</sub>*. This goal is then pursued by the case-based agent in Line 10.

**Table 1: Domination Teams and Descriptions**

Opponent Team	Description	Difficulty
Dom1 Hugger	Sends all agents to domination point 0.	Trivial
First Half Of Dom Points	Sends an agent to the first half +1 domination points. Extra agents patrol between the 2 points.	Easy
2 <sup>nd</sup> Half Of Dom Points	Sends an agent to the second half +1 domination points; extra agents patrol between the two points.	Easy
Each Agent to One Dom	Each agent is assigned to a different domination point And remains there for the entire game.	Medium-easy
Greedy Distance	Each turn the agents are assigned to the closest domination point They do not own.	Hard
Smart Opportunistic	Sends agents to each domination point The team doesn't own. If possible, it will send multiple agents to each un-owned point.	Very hard

**Table 2: Average Percent Normalized Difference in the Game AI System vs. Opponent Scores (with average Scores in parentheses)**

Opponent Team (controls enemies)	Game AI System (controls friendly forces)		
	HTNbots	HTNbots-GDA	CB-gda
Dom1 Hugger	<b>81.2%</b> † (20,002 vs. 3,759)	80.9% (20,001 vs. 3,822)	81.0% (20,001 vs. 3,809)
First Half Of Dom Points	<b>47.6%</b> (20,001 vs. 10,485)	42.0% (20,001 vs. 11,605)	45.0% (20,000 vs. 10,998)
2 <sup>nd</sup> Half Of Dom Points	<b>58.4%</b> (20,003 vs. 8,318)	12.5% (20,001 vs. 17,503)	46.3% (20,001 vs. 10,739)
Each Agent to One Dom	<b>49.0%</b> (20,001 vs. 10,206)	40.6% (20,002 vs. 11,882)	45.4% (20,001 vs. 10,914)
Greedy Distance	-17.0% (16,605 vs. 20,001)	0.4% (19,614 vs. 19,534)	<b>17.57%</b> (20,001 vs. 16,486)
Smart Opportunistic	-19.4% (16,113 vs. 20,001)	-4.8% (19,048 vs. 20,001)	<b>12.32%</b> (20,000 vs. 17,537)

†**Bold face** denotes the highest average measure in each row

## 7 Empirical Study

We performed an exploratory investigation to assess the performance of CB-gda. Our claim is that our case-based approach to GDA can outperform our previous rule-based approach (GDA-HTNbots) and a non-GDA replanning system (HTNbots [13]) in playing DOM games. To assess this hypothesis we used a variety of fixed strategy opponents as benchmarks, as shown in Table 1. These opponents are displayed in order of increasing difficulty.

We recorded and compared the performance of these systems against the same set of hard-coded opponents in games where 20,000 points are needed to win and square maps of size 70 x 70 tiles. The opponents above were taken from course projects and previous research using the DOM game and do not employ CBR or learning. Opponents are named after the strategy they employ. For example, Dom 1 Hugger sends all of its teammates to the first domination point in the map [7]. Our performance metric is the difference in the score between the system and

**Table 3:** Average Percent Normalized Difference in the Game AI System vs. Opponent Scores (with average Scores in parentheses) with Statistical Significance

Opponent	CB-gda – Map 1	CB-gda - Map 2
Dom 1 Hugger	80.8% (20003 vs. 3834)	78.5% (20003 vs. 4298)
	81.2% (20001 vs. 3756)	78.0% (20000 vs. 4396)
	80.7% (20001 vs. 3857)	77.9% (20003 vs. 4424)
	81.6% (20002 vs. 3685)	77.9% (20000 vs. 4438)
	81.0% (20003 vs. 3802)	78.0% (20000 vs. 4382)
<b>Significance</b>	3.78E-11	1.92E-11
First Half of Dom Points	46.0% (20000 vs. 10781)	53.1% (20000 vs. 9375)
	45.8% (20001 vs. 10836)	56.7% (20002 vs. 8660)
	44.9% (20001 vs. 11021)	54.6% (20002 vs. 9089)
	46.1% (20000 vs. 10786)	52.0% (20001 vs. 9603)
	43.4% (20001 vs. 11322)	53.7% (20001 vs. 9254)
<b>Significance</b>	4.98E-08	1.38E-07
Second Half of Dom Points	45.6% (20002 vs. 10889)	60.6% (20000 vs. 7884)
	47.2% (20002 vs. 10560)	61.7% (20000 vs. 7657)
	44.1% (20001 vs. 11188)	61.7% (20000 vs. 7651)
	45.1% (20000 vs. 10987)	61.0% (20001 vs. 7797)
	45.8% (20000 vs. 10849)	60.8% (20002 vs. 7848)
<b>Significance</b>	4.78E-08	7.19E-10
Each Agent to One Dom	46.1% (20001 vs. 10788)	54.9% (20002 vs. 9019)
	46.2% (20000 vs. 10762)	53.7% (20002 vs. 9252)
	44.7% (20002 vs. 11064)	56.8% (20001 vs. 8642)
	44.6% (20000 vs. 11077)	55.4% (20000 vs. 8910)
	47.6% (20002 vs. 10481)	57.7% (20002 vs. 8469)
<b>Significance</b>	6.34E-08	7.08E-08
Greedy Distance	6.4% (20001 vs. 18725)	95.6% (20003 vs. 883)
	8.3% (20001 vs. 18342)	92.7% (20002 vs. 1453)
	5.0% (20000 vs. 18999)	64.6% (20004 vs. 7086)
	9.0% (20001 vs. 18157)	94.9% (20004 vs. 1023)
	12.7% (20001 vs. 17451)	98.0% (20004 vs. 404)
<b>Significance</b>	1.64E-03	6.80E-05
Smart Opportunistic	4.5% (20000 vs. 19102)	13.4% (20001 vs. 17318)
	11.5% (20000 vs. 17693)	13.9% (20001 vs. 17220)
	11.5% (20000 vs. 17693)	1.0% (20001 vs. 19799)
	10.6% (20000 vs. 17878)	10.7% (20002 vs. 17858)
	13.4% (20009 vs. 17333)	12.0% (20003 vs. 17594)
<b>Significance</b>	1.23E-03	1.28E-03

opponent while playing DOM, divided by the system’s score. The experimental setup tested these systems against each of these opponents on the map used in the experiments of GDA-HTNbots [4]. Each game was run three times to account for the randomness introduced by non-deterministic game behaviors. Each bot follows the same finite state machine. Thus, the difference of results is due to the strategy pursued by each team rather than by the individual bot’s performance.

The results are shown in Table 2, where each row displays the normalized average difference in scores (computed over three games) against each opponent. It also shows the average scores for each player. The results for HTNbots and GDA-HTNbots are the same as reported in [4], while the results for CB-gda are new. We repeated the same experiment with a second map and obtained results consistent with the ones presented in Table 2 except for the results against *Greedy*, for which we obtained inconclusive results due to some path-finding issues.

In more detail, we report the results of additional tests here designed to determine whether the performance differences between CB-gda and the opponent team strategies are statistically significant. Table 3 displays the results of playing 10 games over two maps (5 games per map) against the hard-coded opponents. We tested the difference in score between the opponents using the Student’s t-test. For the significance value  $p$  of each opponent, the constraint  $p < 0.05$  holds. Hence, the score difference is statistically significant.

**Table 4:** Average Percent Normalized Difference for the Dynamic Game AI Systems vs. CB-gda Scores (with average scores in parentheses)

Opponent Team (controls enemies)	Game AI System (controls friendly forces)
	CB-gda’s Performance
HTNbots	8.1% (20,000 vs. 18,379)
GDA-HTNbots	23.9% (20,000 vs. 15,215)

We also ran games in which the two dynamic opponents (i.e., HTNbots and GDA-HTNbots) competed directly against CB-gda using the same setup as reported for generating Table 2. As shown in Table 4, CB-gda easily outperformed the other two dynamic opponents. Again, we repeated this study with a second map and obtained results consistent with the ones we present in Table 4.

## 8 Discussion and Future Work

Looking first at Table 2, CB-gda outperformed GDA-HTNbots, alleviating some of the weaknesses that the latter exhibited. Specifically, against the easier and medium difficulty-level opponents (the first 4 in Table 2), HTNbots performed better than GDA-HTNbots (i.e., GDA-HTNbots outperformed those easy opponents but HTNbots did so by a wider margin). The reason for this is that the rule-based GDA strategy didn’t recognize that HTNbots had already created an immediate winning strategy; it should have not suggested alternative goals. CB-gda still suffers from this problem; it suggests new goals even though the case-based agent is winning from the outset. However, CB-gda’s performance is very similar to HTNbots’s performance and only against the third opponent (2<sup>nd</sup> Half Of Dom Points) is HTNbots’s performance observably better. Against the most difficult opponents (the final two in Table 2), GDA-HTNbots outperformed HTNbots, which demonstrated the potential utility of the GDA framework. However, GDA-HTNbots was still beaten by Smart Opportunistic (in contrast, HTNbots did much worse), and it managed to draw against

Greedy (in contrast, HTNbots lost). In contrast, CB-gda clearly outperforms these two opponents and is the only dynamic game AI system to beat the Smart Opportunistic team. Comparing CB-gda to GDA-HTNbots, CB-gda outperformed HTNbots-GDA on all but one opponent, Dom 1 Hugger, and against that opponent the two agents recorded similar score percentages. From Table 4 we observe that CB-gda outperforms both HTNbots and GDA-HTNbots.

One reason for these good results is that CB-gda’s cases were manually populated in PCB and MCB by an expert DOM player. Therefore, they are high in quality. In our future work we want to explore how to automatically acquire the cases in PCB and MCB. Cases in the PCB are triples of the form  $(s_c, g_c, e_c, pl)$ ; these could be automatically captured in the following manner. If the actions in the domain are defined as STRIPS operators (an action is a grounded instance of an operator), then  $e_c$  can be automatically generated by using the operators to find the sequence of actions that achieves  $g_c$  from  $s_c$  (i.e.,  $e_c$  is the observed state of the game after  $g_c$  is satisfied). This sequence of actions will form the plan  $pl$ . Cases in MCB have the form  $(m_c, g_c)$ . These cases can be captured by observing an agent playing the game. The current state  $s_c$  can be observed from the game, and the expectation  $e_c$  can be obtained from PCB. Thus, it is possible to compute their mismatch  $m_c$  automatically.

We plan to investigate the use of reinforcement learning to learn which goal is the best choice instead of a manually-coded case base. Learning the cases could enable the system to learn in increments, which would allow it to address the problem of dynamic planning conditions. We also plan to assess the utility of GDA using a richer representation of the state. As explained earlier, states are currently represented as  $n$ -tuples that denote the owner of each domination point. Thus, the total number of states is  $(t+1)^d$ . In our maps  $d=4$  domination points and there were only two opponents (i.e.,  $t = 2$ ), which translates into only 81 possible states. For this reason, our similarity relations were reduced to equality comparisons. In the future we plan to include other kinds of information in the current state to increase the granularity of the agent’s choices, which will result in a larger state space. For example, if we include information about the locations of CB-gda’s own bots, and the size of the map is  $n \times m$ , then the state space will increase to  $(t+1)^d \times (n \times m)^b$ . In a map where  $n = m = 70$  and the number of bots on CB-gda’s team is 3, then the state space will increase from size 81 to  $81 \times 4900^6$  states. This will require using some other form of state abstraction, because otherwise the size of PCB would be prohibitively long.

We plan to use continuous environment variables and provide the system represent and reason about these variables. The explanation generator aspect of the agent could be expanded to dynamically derive explanations via a comprehensive reasoning mechanism. Also, we would like to incorporate the reasoning that some discrepancies do not require goal formulation.

## 9 Summary

We presented a case-based approach for goal driven autonomy (GDA), a method for reasoning about goals that was recently introduced to address the limitation of classical AI planners, which assume goals are static (i.e., they never change), and cannot reason about nor self-select their goals. In a nutshell, our solution involves maintaining a case base that maps goals to expectations given a certain state (the planning case base - PCB) and a case base that maps mismatches to new goals (the mismatch-goal case base - MCB). We introduced an algorithm that implements the

GDA cycle and uses these case bases to generate new goals dynamically. In tests on playing Domination (DOM) games, the resulting system (CB-gda) outperformed a rule-based variant of GDA (GDA-HTNbots) and a pure replanning agent (HTNbots) against the most difficult manually-created DOM opponents and performed similarly versus the easier ones. In further testing, we found that CB-gda significantly outperformed each of these manually-created DOM opponents. Finally, in direct matches versus GDA-HTNbots and HTNbots, CB-gda outperformed both algorithms.

## Acknowledgements

This work was sponsored by DARPA/IPTO and NSF (#0642882). Thanks to PM Michael Cox for providing motivation and technical direction. The views, opinions, and findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of DARPA or the DoD.

## References

1. Moltke, H.K.B.G. von. Militarische werke. In D.J. Hughes (Ed.) *Moltke on the art of war: Selected writings*. Novato, CA: Presidio Press. (1993)
2. Dearden R., Meuleau N., Ramakrishnan S., Smith, D., & Washington R. Incremental contingency planning. In M. Pistore, H. Geffner, & D. Smith (Eds.) *Planning under Uncertainty and Incomplete Information: Papers from the ICAPS Workshop*. Trento, Italy. (2003)
3. Goldman, R., & Boddy, M. Expressive planning and explicit knowledge. *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*. pp. 110-117. Edinburgh, UK: AAAI Press. (1996)
4. Muñoz-Avila, H., Aha, D.W., Jaidee, U., Klenk, M., & Molineaux, M. Applying goal directed autonomy to a team shooter game. *To appear in Proceedings of the Twenty-Third Florida Artificial Intelligence Research Society Conference*. Daytona Beach, FL: AAAI Press. (2010)
5. Molineaux, M., Klenk, M., & Aha, D.W. *Goal-driven autonomy in a Navy strategy simulation*. *To appear in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Atlanta, GA: AAAI Press. (2010)
6. López de Mantaras, R, McSherry, D., Bridge, D.G., Leake, D.B., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K.D., Keane, M., Aamodt, A., & Watson, I.D. Retrieval, reuse, revision and retention in case-based reasoning. *Knowledge Engineering Review*, **20**(3), 215-240. (2005)
7. Auslander, B., Lee-Urban, S., Hogg, C., & Munoz-Avila, H. Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. *Proceedings of the Ninth European Conference on Case-Based Reasoning*. pp. 59-73. Trier, Germany: Springer. (2008)
8. Nau, D.S. Current trends in automated planning. *AI Magazine*, **28**(4), 43–58. (2007)
9. Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. SHOP: Simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. pp. 968-973. Stockholm: AAAI Press. (1999)
10. Cox, M.T. Perpetual self-aware cognitive agents. *AI Magazine*, **28**(1), 32-45. (2007)
11. Fox, M., Gerevini, A., Long, D., & Serina, I. Plan stability: Replanning versus plan repair. *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*. pp. 212-221. Cumbria, UK: AAAI Press. (2006)

12. Warfield, I., Hogg, C., Lee-Urban, S., Munoz-Avila, H. Adaptation of hierarchical task network plans. *Proceedings of the Twentieth Flairs International Conference* pp. 429-434. Key West, FL: AAAI Press. (2007).
13. Hoang, H., Lee-Urban, S., & Muñoz-Avila, H. Hierarchical plan representations for encoding strategic game AI. *Proceedings of the First Conference on Artificial Intelligence and Interactive Digital Entertainment*. pp. 63-68. Marina del Ray, CA: AAAI Press. (2005)
14. Ayan, N.F., Kuter, U., Yaman F., & Goldman R. Hotride: Hierarchical ordered task replanning in dynamic environments. In F. Ingrand, & K. Rajan (Eds.) *Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop*. Providence, RI. (2007)
15. Myers, K.L. CPEF: A continuous planning and execution framework. *AI Magazine*, **20**(4), 63-69. (1999)
16. Ghallab, M., Nau, D.S., & Traverso, P. *Automated planning: Theory and practice*. San Mateo, CA: Morgan Kaufmann. (2004)
17. van den Briel, M., Sanchez Nigenda, R., Do, M.B., & Kambhampati, S. Effective approaches for partial satisfaction (over-subscription) planning. *Proceedings of the Nineteenth National Conference on Artificial Intelligence*. pp. 562-569. San Jose, CA: AAAI Press. (2004)
18. Coddington, A.M., & Luck, M. Towards motivation-based plan evaluation. *Proceedings of the Sixteenth International FLAIRS Conference*. pp. 298-302. Miami Beach, FL: AAAI Press. (2003)
19. Meneguzzi, F.R., & Luck, M. (2007). Motivations as an abstraction of meta-level reasoning. *Proceedings of the Fifth International Central and Eastern European Conference on Multi-Agent Systems*. pp. 204-214. Leipzig, Germany: Springer.
20. Bergmann, R. *Experience management: Foundations, development methodology, and internet-based applications*. New York: Springer. (2002)