In Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2011), pp. 329--336, Seoul, South Korea, September 2011. IEEE.

UT^2: Human-like Behavior via Neuroevolution of Combat Behavior and Replay of Human Traces

Jacob Schrum, Igor V. Karpov and Risto Miikkulainen

***Abstract*—The UT^2 bot, which had a humanness rating of 27.2727% in BotPrize 2010, is based on two core ideas: (1) multiobjective neuroevolution is used to learn skilled combat behavior, but filters on the available combat actions ensure that the behavior is still human-like despite being evolved for performance, and (2) a database of traces of human play is used to help the bot get unstuck when its navigation capabilities fail. Several changes have recently been made to UT^2: Extra input features have been provided to the bot to help it evolve better combat behavior, the role of human traces in the navigation of the bot has been expanded, and an extra control module has been added which encourages the bot to *observe* other players the way a human would, rather than simply battle them. These changes should make UT^2 act more human-like in this year's BotPrize competition.**

I. BotPrize

The BotPrize competition has been held as part of the Computational Intelligence and Games conference since 2008. The competition is a Turing Test [27] for bots in the game Unreal Tournament 2004 (UT2004), a first-person shooter game in which players run around 3D arenas trying to kill each other for points. The goal of the competition is to see if computer game bots can fool human judges into believing they are human at least 50% of the time.

The first two years of the competition [11] used the standard Turing Test format of one human judge, one computer program, and one human confederate per match. Judges designated one opponent from each match as a human and rated each opponent on a humanness scale. This format was changed for the 2010 competition [12] into a judging game, where judgements are made during the match using a special judging gun. This version of the competition lacks human confederates, and instead puts all bots and all human judges into a match together. Players can judge each opponent once with the judging gun, and the final humanness rating of each player is the percentage of times the player was judged as a human out of all judgments against it. The bots face an extra challenge in this format, because they have access to the judging gun as well. Whenever any player, human or bot, correctly judges an opponent, that opponent dies and the shooter receives 10 points. However, the shooter dies and loses 10 points if the wrong judgment was made.

In the 2010 competition the UT^2 bot took 2nd place with a humanness rating of 27.2727%. A list of the humanness ratings of all participants is shown in Table I. UT^2 is the entry from the University of Texas at Austin, and the name stands for **U**niversity of **T**exas in **U**nreal **T**ournament. This version of the bot, referred to as UT^2-2010 for the rest of this paper, was based on two core ideas: (1) multiobjective neuroevolution was used to learn skilled combat behavior, but filters on the available combat actions ensured that the behavior was still human-like despite being evolved for performance, and (2) a database of traces of human play was used to help the bot get unstuck when its navigation capabilities failed. UT^2-2010 is described in full detail in upcoming chapters [16, 20] for the book Believable Bots.

Player	Type	Humanness
Mads Frost	Human	80.0000%
Simon and Will Lucas	Human	59.0909%
Ben Weber	Human	48.2759%
Nicola Beume	Human	47.0588%
Minh Tran	Human	42.3077%
Gordon Calleja	Human	38.0952%
Mike Preuss	Human	35.4839%
Conscious-Robots	Bot	31.8182%
UT^2	Bot	27.2727%
ICE-2010	Bot	23.3333%
Discordia	Bot	17.7778%
w00t	Bot	9.3023%

TABLE I: BotPrize 2010 Results (UT^2 highlighted). Humanness equals the number of human judgments divided by the total judgments, all multiplied by 100. UT^2 beat three entries to get 2nd place. However, none of the entries are more human than the least human of humans.

The UT^2 bot has been modified in several ways since 2010 in order to increase its humanness rating for this year's competition: Extra input features have been provided to help it evolve better combat behavior, extra filters on combat actions inject more human knowledge into the bot, the role of human traces in the navigation of the bot has been expanded, and an extra control module has been added which encourages the bot to *observe* other players the way a human would, rather than simply battle them. The remainder of this chapter first discusses the general architecture used to control UT^2, and then goes into detail describing each of these enhancements. These changes were developed with the rules from the 2010 competition in mind, and even though the rules for the upcoming 2011 competition are still being worked out, the changes made to UT^2 should improve its performance in this year's BotPrize competition.

II. Architecture

The UT^2 bot uses a behavior-based architecture in which a list of behavior modules is cycled through in priority order on every logic cycle. This architecture is somewhat similar

Jacob Schrum, Igor V. Karpov and Risto Miikkulainen are with the Department of Computer Science, University of Texas at Austin, Austin, TX 78712 USA (email: {schrum2,ikarpov,risto}@cs.utexas.edu)

to both the POSH framework [5], and to behavior trees [14]. Each behavior module has a trigger, and if a module's trigger fires on a given cycle, then that module defines the behavior of the agent for that logic cycle. The full architecture is shown in Fig. 1. The control modules, from highest to lowest execution priority, are:

1) *UNSTUCK*: Getting unstuck has highest priority since being stuck is a very bot-like behavior that prevents the execution of other actions.
2) *GET DROPPED WEAPON*: Causes the bot to rush and pick up weapons that enemies drop upon dying.
3) *IMPORTANT ITEM*: Makes the bot pursue items like the Keg o' Health (which allows players to exceed the normal maximum health limit) and UDamage (which doubles the damage dealt by the bot for 30 seconds) whenever they are nearby, even if it means breaking off from combat. Obtaining these items is considered more important than fighting.
4) *GET GOOD WEAPON*: Whenever the bot has only the starting weapons, it is not able to put up much of a fight. Therefore, running to get a good weapon is more important than fighting.
5) *JUDGE*: The judging gun has infinite ammo, so it is possible to judge at any time. This module decides when to judge a given opponent, and whether the opponent should be judged as a human or bot.
6) *OBSERVE*: In the judging game variant of UT2004 used for BotPrize, it is very common for humans to stand back and watch other players, particularly groups of opponents. This module tries to emulate such observation behavior.
7) *SHIELD GUN*: The shield gun is a melee weapon with regenerating ammo that is very hard to use. It also allows players to shield themselves from incoming projectiles. However, despite its versatility, it is harder to use than the standard projectile weapons available. Therefore, the bot only uses the shield gun if it is out of ammo for all other weapons, and has no reason to judge or observe.
8) *BATTLE*: If the bot does have ammo when encountering an enemy that it neither wants to judge nor observe, then it enters combat.
9) *CHASE*: If an opponent is lost during combat, the bot will chase after it.
10) *RETRACE*: If there are no interesting items or opponents to interact with, the bot simply explores the level. This module uses replay of human traces to explore the level, when the traces are available.
11) *PATH*: As a failsafe for when human trace data is not available, the bot can explore using the level's built-in navigation graph.

In terms of Computational Intelligence, the two most interesting features of `UT^2` are that its combat behavior is defined via an evolved neural network (in *BATTLE*), and both its navigation and its routine for getting unstuck make use of a database of traces of human behavior (via the `Human Retrace Controller` from both *UNSTUCK* and *RETRACE*). Therefore, these modules will be discussed in detail next. Later, the *OBSERVE* module will also be discussed, because this is a new module added for the 2011 competition.

III. Evolved Battle Controller

`UT^2-2010`'s `Battle Controller` was learned using multiobjective constructive neuroevolution. This method of neuroevolution is the same as in [21], and uses the popular multiobjective evolutionary algorithm Non-Dominated Sorting Genetic Algorithm II (NSGA-II [8]) in conjunction with neuroevolution principles inspired by Neuro-Evolution of Augmenting Topologies (NEAT [22]).

A. Neuroevolution

Neuroevolution is the application of evolution to artificial neural networks. `UT^2`'s combat behavior was learned via constructive neuroevolution, meaning that the networks start with minimal structure and only become more complex as a result of mutations across several generations. The initial population of networks consists of individuals with no hidden layers, i.e. only input and output nodes. Furthermore, these networks are sparsely connected in a style similar to Feature Selective NEAT (FS-NEAT [31]). Initializing the networks in this way allows them to easily ignore any inputs that are not, or at least not *yet*, useful. `UT^2` makes use of a large number of network inputs, so it is important to be able to ignore certain inputs early in evolution, when establishing a baseline policy is more important than refining the policy. The full set of inputs used by `UT^2-2010` is described in [20], but some extra inputs added to `UT^2` for this year's competition are described later in section III-C.

Three mutation operators were used to change network behavior. The weight mutation perturbs the weights of existing network connections, the link mutation adds new (potentially recurrent) connections between existing nodes, and the node mutation splices new nodes along existing connections. Recurrent connections transmit signals that are not processed by the network until the following time step, which makes them particularly useful in partially observable domains. In the context of reinforcement learning problems [25], such as UT2004, an environment is partially observable if the current observed state cannot be distinguished from other observed states without memory of past states. Recurrent connections help in these situations because they encode and transmit memory of past states. These mutation operators are similar to those used in NEAT [22]. Crossover was not used.

This section explained the representation that was used to evolve policies for `UT^2`. The next section explains the algorithm controlling how the space of policies was searched.

B. Evolutionary Multiobjective Optimization

In multiobjective optimization, two or more conflicting objectives are optimized simultaneously. A multiobjective approach is important for domains like UT2004, which involve many conflicting objectives: kill opponents, avoid

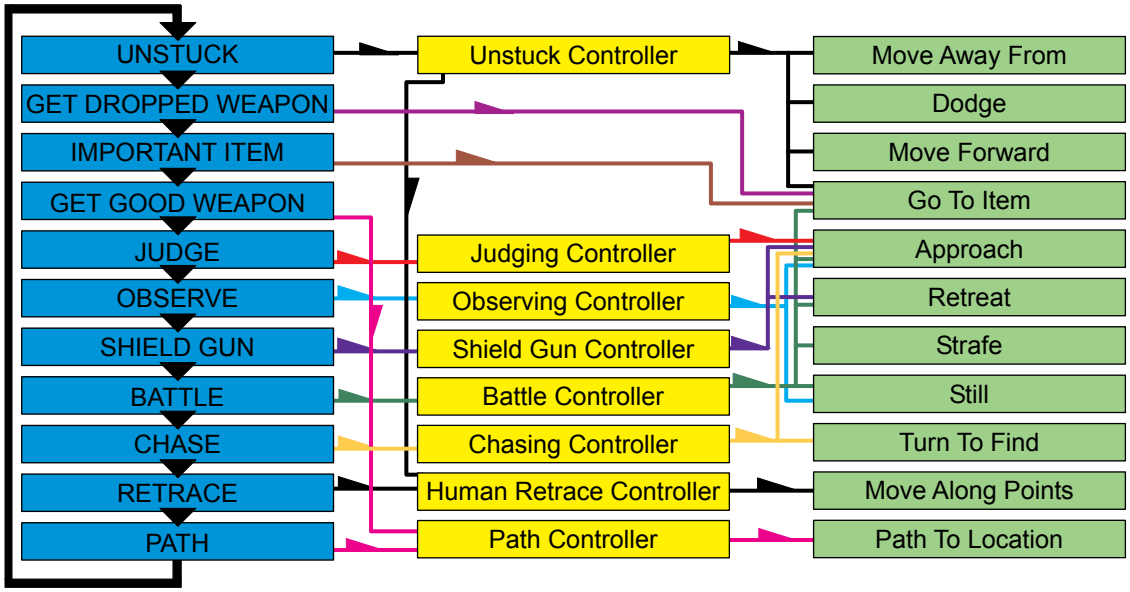


Fig. 1: `UT^2` Architecture. Control cycles through modules listed (on the left) once every logic cycle. If a module's trigger fires, then that module will define the bot's next action. Most modules have an associated controller (middle column) that further arbitrates between several available actions, or otherwise aggregates and makes use of information relevant to the actions performed by that module. All actions available to the controllers are in the right column. One of these actions is executed each logic cycle. Some control modules are simple enough that they do not need controllers: They carry out a specific action directly. Most of the control in this diagram flows from left to right, but note that the `Unstuck Controller` can actually make use of the `Human Retrace Controller` to define its action. This behavior-based architecture modularizes bot behaviors, making the overall behavior easier to understand, and making programming and troubleshooting easier.

damage, etc. Important concepts in dealing with multiple objectives are Pareto dominance and optimality. [1]

Pareto Dominance: Vector $\vec{v} = (v_1, \ldots, v_n)$ dominates $\vec{u} = (u_1, \ldots, u_n)$, i.e. $\vec{v} \succ \vec{u}$, iff

1. $\forall i \in \{1, \ldots, n\} : v_i \geq u_i$, and
2. $\exists i \in \{1, \ldots, n\} : v_i > u_i$.

Pareto Optimality: A set of points $\mathcal{A} \subseteq \mathcal{F}$ is Pareto optimal iff it contains all points such that $\forall \vec{x} \in \mathcal{A}$: $\neg\exists \vec{y} \in \mathcal{F}$ such that $\vec{y} \succ \vec{x}$. The points in $\mathcal{A}$ are non-dominated, and make up the non-dominated Pareto front of $\mathcal{F}$.

The above definitions indicate that one solution is better than (i.e. dominates) another solution if it is strictly better in at least one objective and no worse in the others. The best solutions are not dominated by any other solutions, and make up the Pareto front of the search space. Therefore, solving a multiobjective optimization problem involves approximating the Pareto front as best as possible, which is exactly what Evolutionary Multiobjective Optimization (EMO) methods do. The EMO method used in this work is NSGA-II [8].

NSGA-II uses a $(\mu+\lambda)$ selection strategy. In this paradigm, a parent population of size μ is evaluated, and then used to produce a child population of size λ. Selection is performed on the combined parent and child population to give rise to a new parent population of size μ. NSGA-II uses $\mu = \lambda$.

NSGA-II sorts the population into non-dominated layers in terms of each individual's fitness scores. For a given population, the first non-dominated layer is simply the Pareto front of that population. If this first layer is removed, then the second layer is the Pareto front of the remaining population. By removing layers and recalculating the Pareto front, the whole population can be sorted. Elitist selection favors individuals in the less-dominated layers. Within the same layer, individuals that are more distant from others in objective space are preferred based on a metric called crowding distance. The crowding distance metric ensures the exploration of diverse trade-offs between objectives.

Applying NSGA-II to a problem results in a population containing a close approximation to the true Pareto front with individuals spread out evenly across the trade-off surface between objectives. The details of how this process was carried out in UT2004 are covered in the next section.

C. *Evolution of* `UT^2`

This year a new controller was evolved using the methods described above. Evolution occurred against native UT2004 bots in the relatively small map `DM-1on1-Albatross` to ensure that as much of the bot's time was spent in combat as possible. The set of objectives used was a simple set of three: damage dealt (maximize), damage received (minimize), and number of collision events with level geometry (minimize). In a game as complex as UT2004, there are many other sensible objectives that could be used, such as maximizing accuracy and death match score, but only three objectives were used because NSGA-II tends to have trouble making useful distinctions between solutions as the number of objectives grows.

[1] These definitions assume a maximization problem. Objectives that are to be minimized can simply have their values multiplied by -1.

Several input sensors were added to the bot's neural network, most interesting of which are the opponent-movement sensors. These are sensors that determine heuristically whether an opponent is executing one of a small set of combat movement options that are available to the bot. Such sensors were built to make evolution of mimicry possible should it prove useful in combat. Such mimicry should be useful since evolved behavior is generally not inherently human-like.

However, human-like tendencies are enforced in how the outputs of the network are interpreted. As with UT^2-2010, the combat actions available to UT^2 are defined relative to its current opponent, which was selected via a scripted routine. The available actions are: Approach, Retreat, Strafe (left or right), stand Still, and Go To Item which is nearest. The bot always looks at the opponent while performing these actions, and thus seems to be focused in a human-like manner. The opponent-movement sensors indicate if the bot's opponent is performing any of these actions. During these actions, the bot has the option of jumping and/or shooting. UT^2-2010 evolved the decision of when to shoot, but this year's version simply favors shooting whenever a target is available. Many new filters and restrictions on when these actions can be performed have also been added, as described in the following list:

- Disallow Go To Item if
  - Item undesirable.
- Disallow Retreat if
  - Wall directly behind bot.
- Disallow Approach if
  - Within range with sniping weapon, or
  - Opponent not retreating and within range with rocket launcher, or
  - Very close to opponent.
- Disallow Strafe if
  - Extremely close to opponent.
- Disallow Still if
  - Threatened at close range, or
  - Bot is far and weapon is only effective when close.
- Force Approach if
  - Charging secondary Bio-Rifle shot.
- Force Strafe if
  - Wall is close on opposite side.
- Force Still if
  - Bot is either not Approaching, has the high ground, or is not using a close-range weapon, and
    - Not threatened when using medium-range weapon, or
    - Not threatened when using close-range weapon from ideal distance, or
    - Using sniping weapon from afar or high ground.

In the hierarchical list above, an action that is disallowed can no longer be forced. These and other restrictions on bot behavior were added based on the authors' knowledge of what humans consider to be human-like/bot-like behavior in UT2004. Additionally, the bot's accuracy is reduced with respect to how much it and its target are moving, as well as the distance between the two individuals, in order to make the bot fallible in a human-like manner.

Another important change in how the Battle Controller is used relates to use of the judging gun. UT^2-2010's Judging Controller made use of the Battle Controller to define bot movement while attempting to judge opponents. However, competition experience has shown that the power and importance of judging makes getting a successful judgement important enough that human players are less concerned with maneuvering to avoid damage than usual. Therefore, UT^2's Judging Controller does not use the Battle Controller to select from all available combat movement actions, but instead simply chooses the Approach command every time. This restriction makes the bot appear more focused when attempting to judge an opponent.

IV. Human Trace Replay

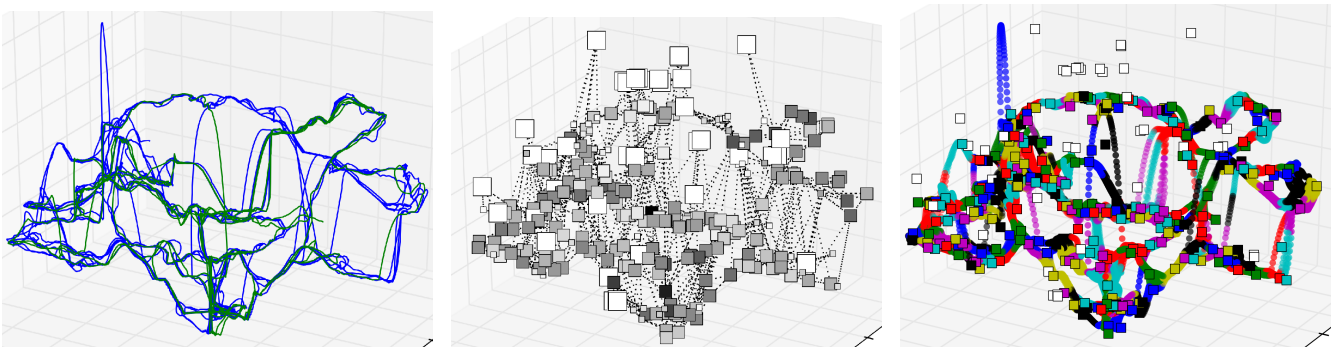
UT^2-2010 made use of human traces purely for the purpose of getting the bot unstuck whenever one of several stuck triggers fired, thus indicating that navigation had somehow failed. The current version of the bot was modified in several ways: (1) the Human Retrace Controller was improved to play back long traces more reliably, (2) the bot was supplied with improved and filtered databases of recorded human behavior for each level to be used in the competition, (3) the UNSTUCK module now combines human trace behavior with scripted actions, and (4) the new RETRACE module uses prolonged human trace replay in order to explore levels in the absence of other goals.

A. Storing Human Data

There is a separate database of recorded human traces corresponding to each level in UT2004 (Fig. 2). Each database consists of sequences of agent locations stored along with the game time that the player was at that position. The sequence of locations for one player ordered by time represents a trace of how a human player moved through a given level.

In order to make the sequences stored in the database smooth and useful for getting unstuck and wandering around the levels, the data recorded from the human players is filtered. Sequences are broken up into separate subsequences if they are broken up by death, or by space or time discontinuities. The threshold values for deciding these were determined experimentally and held fixed during competition and evaluation. There is a trade-off between the average length of sequences in the database and their continuity, and sequences that were too short were removed from the database after it was broken up into appropriate subsequences.

However, in order to ensure that this filtering process would produce usable data, traces were collected in a synthetic manner: Individual players ran around levels by themselves with no enemies, with the purpose of collecting items while exploring the level as much as possible. Such



(a) Two Human Traces (b) Navigation Graph (c) Trace-Navpoint Mapping

Fig. 2: Human trace data for DM-Antalus. The human trace data used for the UT2004 level DM-Antalus are plotted in three dimensions, X, Y and Z. The original pose traces (2a) are overlayed onto the navigation graph (2b) and indexed by the nearest vertex/navpoint for faster retrieval (2c). In Fig. 2b, the relative size and shading of the squares at the vertices of the navigation graph indicates the number of points in the database in each vertex' neighborhood - a measure of coverage used during quality control. In Fig. 2c (color figure), the pose data are colored according to the nearest navpoint to show the Voronoi region indexing scheme used during retrieval.

synthetic data is free of the erratic movement which is characteristic of combat, but which would look strange if replayed without an opponent.

The locations within the traces are indexed by their nearest navpoint within the level, as in Voronoi cells [9, 29]. Each such set of locations is then stored in a KD-tree data structure to support fast ($O(\log(n))$) nearest neighbor queries [4]. This indexing scheme speeds up the operation of finding the nearest point of the nearest trace when needed.

B. Replaying Human Traces

Whenever a trace is retrieved for replay, the bot picks points along the trace starting from near its current location, and uses the Move Along Points action to move directly to the first point while planning ahead to the next point in the sequence. The points on the selected path are picked according to an estimate of the distance that the bot covers during the course of a single decision frame, and this estimate is continuously updated as the bot plays back human traces. Such adaptive planning results in smoother, more human-like movement than simply moving through the sequence of points from a given trace.

The quality of playback of human traces depends on a number of factors, including level geometry (open vs. enclosed spaces, number of turns, slopes), the nature of the human data (number and frequency of jumps and stops, autonomous movement vs. movement due to combat), and the fidelity of the estimated distance covered used in trace playback. It is therefore important to control the quality of the resulting behavior. Such quality control was achieved by manually observing the retrace behavior in isolation, by collecting and analyzing statistics about the actions and conditions occurring during repeated runs, and by visualizing the underlying data in meaningful ways.

C. Getting Unstuck

The UT^2-2010 bot made use of human traces exclusively for getting unstuck. The current version of the bot still uses human traces for this purpose, but also uses scripted actions for getting unstuck under specific circumstances. The scripted responses to getting stuck are to Move Forward if standing still, Move Away from walls and agents with which the bot is colliding, and to Dodge away from obstacles if collisions are occurring with high frequency. Dodge is also used to escape when the bot gets stuck under moving elevators. Elevators can be particularly confusing because they contain moving navpoints that may not be where the bot expects them to be.

These scripted responses usually work well in these situations, but if these responses fail repeatedly, or if the bot is near the same navpoint for too long, or if the bot finds itself significantly removed from any navpoint in the navigation graph, then the bot will try using human traces to get unstuck.

If there is no reasonably close human trace available, or if the human traces have repeatedly failed to get the bot unstuck, then the bot resorts to random unstuck actions, which include Move Forward, Move Away From, Dodge, and Go To Item.

D. Prolonged Human Retrace

In addition to using human traces to get unstuck, the new control module RETRACE is entirely based on the prolonged playback of human traces, in order to achieve smooth, human-like exploration of the level. Playing back human traces via the RETRACE module results in smoother movement than the lower priority, A*-based PATH module, which is a slightly improved version of the path navigation module used by UT^2-2010. The human data used by RETRACE plays back smoothly because it was created in a synthetic manner, as explained in section IV-A.

E. Measuring Contribution of Human Traces

In order to measure the contribution of the two kinds of human trace playback to the overall quality of the bot's behavior, both qualitative and quantitative observations of the bot were performed under four different conditions:

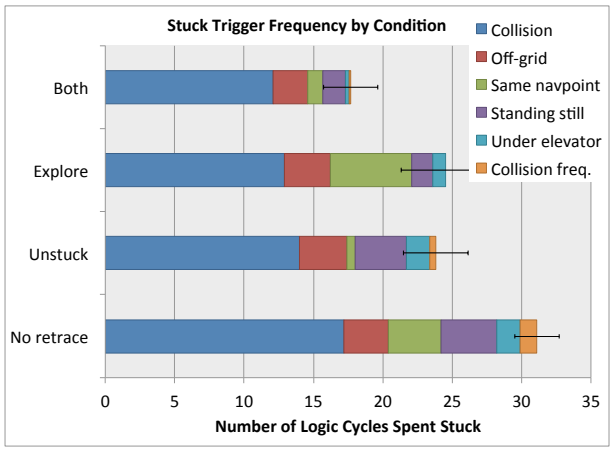


Fig. 3: Average number of frames stuck by condition (color). An average of 10 runs with their standard error bars is shown. Using human traces for exploration, to get unstuck, or for both allows the bot to remain stuck for fewer frames and looks more human-like upon observation.

1) *No retrace*: bot uses no human traces at all.
2) *Unstuck*: bot uses traces only for getting unstuck.
3) *Explore*: bot uses traces only for exploring the level.
4) *Both*: bot uses traces both to explore and to get unstuck.

The results of the comparison in terms of the number and kind of stuck conditions for a typical level are shown in Fig. 3. The amount of time the bot spent stuck was generally smallest when using human traces both for getting unstuck and for exploring the level, and the behavior looked qualitatively smoothest and most human-like when observing the bot in this condition, supporting the final design decision for the competition.

V. Observing While Playing

The `RETRACE` module is a new module that is primarily used when the bot is not interacting with opponents, because it cannot see them. In contrast, the `OBSERVE` module is a new module that is used when the bot is not interacting with opponents that it does see. The `OBSERVE` module models a human's tendency, within the context of the judging game, to stand outside of the action and attempt to figure out whether a given opponent is a human or a bot.

The need for an `OBSERVE` module was overlooked in `UT^2-2010` because 2010 was the first year that BotPrize used the judging game rules. In a regular UT2004 match, there is little reason to simply watch other players, because there are no points to be gained from not engaging in combat. One player might avoid another in order to get more health or a better weapon, but staying near enough to an opponent to watch without actually fighting makes little sense. In contrast, the goal of making correct judgments in BotPrize makes time spent observing worthwhile.

The `Observing Controller` of `UT^2` works as follows: the `OBSERVE` module will take control if the bot sees an opponent or opponents that it has not yet judged, and if it does not perceive itself to be threatened by any of these opponents. The rational behind this triggering mechanism is that humans are only interested in observing individuals they have not yet judged (a visual cue reminds humans whether or not they have judged a given opponent), but passive observing only makes sense if the observer is not actively threatened. Once the bot feels threatened, it will launch into combat. However, if the observed opponent(s) continue to ignore the bot, its desire to judge the observed opponent increases, until a threshold is passed that causes it to commit to judging the observed opponent. This transition is logical, since the purpose for observing is to gain enough information to make a judgment.

The actual actions performed while observing are fairly simple: The bot will stay still if it is close enough to observe the action, and it will approach the observed player if it gets too far away. Though reasonable, this set of actions is fairly simple, and may be improved upon based on observations of games between humans and `UT^2`.

VI. Related Work

The approach presented in this paper combines a modular architecture, multiobjective neuroevolution techniques and the use of recorded human traces in order to create a human-like bot for a commercial first-person shooter game. Several relevant threads of existing literature demonstrate the power of these individual approaches as well as the opportunity to use and to study them further.

A. Bot Evolution

Other researchers have evolved bots in UT2004 and similar first-person shooter games for the sake of maximizing performance. Because these games are so complex, much work has focused on learning some isolated component of good behavior.

For example, Graham et al. evolved artificial neural networks for the task of pathfinding in the game Quake [10], and Karpov et al. evolved similar pathfinding behavior in the original Unreal Tournament [15]. Cuadrado and Saez evolved dodging rules for a bot in UT2004 [7], and Westra and Dignum evolved weapon and item selection behavior for bots in Quake III Arena [30].

Other researchers have focused on evolving complete agents for these challenging games. A common approach to evolving a complete controller for such complex games is to evolve separate component controllers and combine them. This approach was taken by Zanetti and El Rhalibi, who evolved components of a bot controller for Quake III Arena to match target data collected from human experts [32]. Another approach to evolving a complete bot was taken by van Hoorn et al., who evolved a master controller on top of subcontrollers evolved for particular tasks in order to learn the behavior for a bot in UT2004 [28].

These works have been reasonably successful in evolving skilled bot behavior in UT2004 and similar games. However, even when researchers have made use of human data to evolve skilled behavior, the final evaluation has always been in terms of raw performance. The interest in domain performance makes such previous work fundamentally different

from the work presented in this paper, which is primarily concerned with looking human rather than in performing well. In fact, judges' comments from previous competitions have indicated that being too skilled in the domain of UT2004 can result in a player being more likely to be judged as a bot. The need to be skilled within the limits of human performance is why UT^2 uses various filtering mechanisms to constrain evolved combat behavior. Such constraints are a departure from previous work, and are necessary to prevent bots from becoming skilled at the expense of looking human.

B. Human Traces

The use of human player data recorded from games in order to create realistic game characters is a promising direction of research because it can be applied both to games and to the wider field of autonomous agent behavior. This approach is closely related to the concept of Imitation Learning or Learning from Demonstration, especially when expanded to generalize to unseen data [2, 3, 19].

In games, imitation of human traces has previously been used to synthesize movements in the game Quake [26], however this approach has not been evaluated in the framework of a human-like bot competition. The use of trajectory libraries was introduced for developing autonomous agent control policies and for transfer learning [23, 24]. Predictive models of player behavior learned from large databases of human gameplay have been used for multi-agent opponent modeling in the context of the first person shooter Half Life 2 [13]. Imitation learning using supervised models of human drivers was used in order to train agent drivers in the TORCS racing simulator [6]. In robotics, imitation learning approaches have been shown effective as well, for example in task learning and programming robosoccer players in simulation [1]. Statistical analysis of player trajectories was used in order to detect game bots in the first person shooter Quake [18]. Most recently, a competitor team, ICE, is using an interface for creating custom recordings of human behavior in the UT2004 [17].

While human behavior traces and learning from demonstration techniques are finding increasing use in both games and robotics applications, the BotPrize competition offers a unique opportunity to test such methods in creating human-like behavior directly. The challenge of combining imitation and demonstration methods with other types of policy design methods remains to be met.

VII. Future Work

The changes made to UT^2 should improve its chances in the upcoming BotPrize competition. However, there is still work to be done, both to improve the methods used by UT^2 and to understand what it means for a bot to behave in a human-like manner.

The ways in which human traces are currently used are all dependent on having data from the particular level on which the bot will be used. This is a major limitation for a game like UT2004, which allows users to create their own levels, none of which could be properly played by UT^2 without first collecting level-specific human data. Therefore, an interesting direction for future research is to use human data from some levels to train a controller that generalizes to new levels. Supervised learning methods such as artificial neural networks can be used to build generative models of human behavior in games, mapping from a local, egocentric representation of the player's game state to possible future actions. Such models could then be used for direct control as described in this paper or in other ways, for example in order to predict opponent behavior or to rank evolving policies according to a computable "humanness" rating.

As stated above, the actions performed by the Observing Controller are fairly simple. It is currently unclear what humans are actually doing when they observe other players in the UT2004 judging game. Although comments collected from judges in previous competitions have given some data on what they are looking for in human-like behavior, it is still not clear what information they are gaining from each individual interaction with an opponent. The demo files from BotPrize 2010[2] are full of judgments that are difficult for an outside observer (as opposed to the interacting observers that were judges in the competition) to understand. Because the BotPrize competitions have been more concerned with evaluating the humanness of bots than understanding what makes behavior human-like or bot-like, there has been little progress towards attaining the 50% humanness rating required to win the major prize for the competition. It would likely be very informative to run a tournament in which judges are debriefed after each match, and taken through a video of their play while being asked what they were thinking while making each judgment. Such a study could be a major step towards understanding what separates the humans from the bots in UT2004.

VIII. Conclusion

Starting from the UT^2-2010 bot, an improved version of UT^2 has been designed to compete in BotPrize 2011. This version of the bot uses a new neural network to control its combat behavior, evolved with a more streamlined set of objectives in a more combat-intensive scenario against the challenging native UT2004 bots. The bot also makes more extensive use of human traces as part of the new RETRACE module, which makes the bot explore the levels the way humans do. The unstuck behavior of the bot still uses human traces as well, but now also makes use of scripted routines that are more applicable to particular situations. Finally, the bot has a new control module dedicated to observing opponents, which is an important human behavior in the context of BotPrize's judging game. All of these changes should help UT^2 behave in a more human-like manner for BotPrize 2011.

Acknowledgment

This research was supported in part by the NSF under grants DBI-0939454 and IIS-0915038.

[2]http://www.botprize.org/result.html

REFERENCES

[1] R. Aler, J. M. Valls, D. Camacho, and A. Lopez. Programming Robosoccer Agents by Modeling Human Behavior. *Expert Systems with Applications*, 36(2, Part 1):1850–1859, 2009.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A Survey of Robot Learning From Demonstration. *Robotics and Autonomous Systems*, 57(5), 2009.

[3] C. G. Atkeson and S. Schaal. Robot Learning From Demonstration. In *International Conference on Machine Learning*, pages 12–20, 1997.

[4] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] J. J. Bryson. *Intelligence by Design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agents*. PhD thesis, Massachusetts Institute of Technology, 2001.

[6] L. Cardamone, D. Loiacono, and P. L. Lanzi. Learning Drivers for TORCS through Imitation Using Supervised Methods. In *Computational Intelligence and Games*, 2009.

[7] D. Cuadrado and Y. Saez. Chuck Norris Rocks! In *Computational Intelligence and Games*, Sep. 2009.

[8] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. *PPSN VI*, pages 849–858, 2000.

[9] G. L. Dirichlet. Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. *Journal für die reine und angewandte Mathematik*, 40:209–227, 1850.

[10] R. Graham, H. McCabe, and S. Sheridan. Realistic Agent Movement in Dynamic Game Environments. In *Changing Views: Worlds in Play: Digital Games Research Association Conference*, Vancouver, June 2005.

[11] P. Hingston. A Turing Test for Computer Game Bots. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(3):169–186, Sep. 2009.

[12] P. Hingston. A New Design for a Turing Test for Bots. In *Computational Intelligence and Games*, 2010.

[13] S. Hladky and V. Bulitko. An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games. In *Computational Intelligence and Games*, Perth, Australia, 2008.

[14] D. Isla. Managing Complexity in the Halo 2 AI System. In *Game Developers Conference*, 2005.

[15] I. Karpov, T. D'Silva, C. Varrichio, K. O. Stanley, and R. Miikkulainen. Integration and Evaluation of Exploration-Based Learning in Games. In *Computational Intelligence and Games*, 2006.

[16] I. V. Karpov, J. Schrum, and R. Miikkulainen. Believable Bot Navigation via Playback of Human Traces. In P. F. Hingston, editor, *Believable Bots*. Springer, 2011. (To Appear).

[17] S. Murakami, T. Sato, A. Kojima, D. Hirono, N. Kusumoto, and R. Thawonmas. Outline of ICE-CEC2011 and Its Mechanism for Learning FPS Tactics. Extended Abstract for the Human-like Bot Workshop at CEC 2011, June 2011.

[18] H.-K. Pao, K.-T. Chen, and H.-C. Chang. Game Bot Detection via Avatar Trajectory Analysis. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(3):162–175, 2010.

[19] S. Schaal. Learning From Demonstration. In *Advances in Neural Information Processing Systems 9*, 1997.

[20] J. Schrum, I. V. Karpov, and R. Miikkulainen. Human-like Combat Behavior via Multiobjective Neuroevolution. In P. F. Hingston, editor, *Believable Bots*. Springer, 2011. (To Appear).

[21] J. Schrum and R. Miikkulainen. Evolving Agent Behavior In Multiobjective Domains Using Fitness-Based Shaping. In *Genetic and Evolutionary Computation Conference*, July 2010.

[22] K. O. Stanley and R. Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, 10:99–127, 2002.

[23] M. Stolle and C. G. Atkeson. Policies Based on Trajectory Libraries. In *International Conference on Robotics and Automation*, 2006.

[24] M. Stolle, H. Tappeiner, J. Chestnutt, and C. G. Atkeson. Transfer of Policies Based on Trajectory Libraries. In *International Conference on Intelligent Robots and Systems*, 2007.

[25] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[26] C. Thurau, C. Bauckhage, and G. Sagerer. Synthesizing Movements for Computer Game Characters. *Adaptive Behavior*, 3175:179–186, 2004.

[27] A. M. Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, 1950.

[28] N. van Hoorn, J. Togelius, and J. Schmidhuber. Hierarchical Controller Learning in a First-Person Shooter. In *Computational Intelligence and Games*, 2009.

[29] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die reine und angewandte Mathematik*, 133, 1907.

[30] J. Westra and F. Dignum. Evolutionary Neural Networks for Non-Player Characters in Quake III. In *Computational Intelligence and Games*, Sep. 2009.

[31] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl. Automatic Feature Selection in Neuroevolution. In *Genetic and Evolutionary Computation Conference*, 2005.

[32] S. Zanetti and A. E. Rhalibi. Machine Learning Techniques for FPS in Q3. In *ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACE '04, New York, NY, USA, 2004.