Creating Interactive Characters with BDI Agents

Emma Norling
Department of Computer Science
and Software Engineering
The University of Melbourne, 3010
Australia
norling@acm.org

Liz Sonenberg
Department of Information Systems
The University of Melbourne, 3010
Australia
l.sonenberg@unimelb.edu.au

ABSTRACT

This paper discusses the use of BDI agents for the development of human-like synthetic characters. The folk psychological roots of the paradigm map closely to the way people typically explain both their behaviour and that of others, and this greatly facilitates knowledge elicitation and representation. This is illustrated through some examples from a project in which models of expert players of Quake 2 were developed. The knowledge elicitation methodology that was used is explained, and samples of the code are presented, demonstrating the way in which a BDI-based agent programming language can clearly and succinctly capture individual differences. The example presented is of modelling expert players in an existing game, but the paper argues that the same techniques can be used to build a completely original character, using a role-player as the basis. Finally, some of the limitations of the BDI paradigm are examined, with a brief discussion of how they can be addressed, using the existing framework as a basis.

Keywords

User modelling, synthetic characters, BDI agents

1. INTRODUCTION

The use of BDI agents for modelling human behaviour has a considerable history, mainly in military simulation [4, 8, 11, 14]. In the entertainment industry, there is one notable use of the technology – *Black&White* [10] – to create synthetic characters. This paper argues that the BDI agent paradigm, although originally developed for other purposes, is particularly suited to the creation of synthetic characters, particularly those with human characteristics. The actions of characters built using this paradigm arise from the instantiation of partially specified plans, selected to fulfil the character's goals given the beliefs that it has at that point in time. The details of the plan are filled in as the plan progresses, which allows the character designer to generate a wide range of possible courses of action without having to fully specify the details in each path. It also encodes this information in a way that maps closely to the way people describe their reasoning, making it relatively easy to capture the knowledge of non-programming experts, such as actors, role-players or players of the game. While the complexity of the paradigm would (at this stage) limit the number of such agents that could be supported in a single game, it offers considerable opportunities for the modelling of key characters in the game (as in *Black&White*), or, in combination with level-of-detail shifting techniques, to add colour and depth to minor characters during key interactions.

The BDI agent paradigm was originally developed to achieve a balance between reactive and deliberative behaviour [2], rather than for human modelling. However the folk psychological primitives used in the paradigm – beliefs, desires (or goals) and intentions – not only achieve the balance between reactiveness and deliberation that was the original raison d'être of the paradigm, but also correspond to the way people typically explain both their own reasoning process, and that of others. For this reason, the paradigm is well-suited to developing characters which are expected to display a similar level of intelligence to people, and use a similar underlying reasoning process.

We illustrate our case in this paper using examples from a project that involved building models of expert Quake 2 players as 'bots' that could play the deathmatch version of the game against other players. Although in this case the models were of experienced players in an existing game, the techniques used could also be applied to role-players envisioning a character in a game under development. The close mapping between the concepts that the experts used to explain their reasoning/behaviour and the constructs within the BDI paradigm greatly facilitated the knowledge elicitation and model building process. Moreover, the just-in-time planning used by BDI agents (explained in Sect. 2) allows a wide range of complex behaviours *without* having to fully specify every possible course of action. Some illustrations of the knowledge capture and representation process are presented in Sect. 4, using the JACK Intelligent Agents programming language [5], which is explained where necessary.

While the folk psychological roots of the paradigm do lend it to building human-like characters, they also provide a level of abstraction in the reasoning that is not always advantageous. There are many characteristics of human-like behaviour that are not explicitly captured in the paradigm, characteristics which sometimes should have considerable impact on behaviour. Fortunately, although the paradigm does not explicitly represent such characteristics, it is highly amenable to the types of extensions needed to address these issues.

In the next section, we explain the underlying concepts of the BDI paradigm. This is followed in Sect. 3 by a description of the knowledge elicitation process that was used to gather the plans and knowledge used by three expert Quake2 players, the subjects of the experiments. Section 4 then discusses how this knowledge was encoded using JACK, demonstrating the way this process is facilitated by the BDI

paradigm, but also highlight some of the ways in which JACK diverges from a "pure" BDI language. In Sect. 5 we then look at some of the shortcomings of a BDI approach to human modelling, and briefly outline how the BDI paradigm can be extended to address these issues.

2. THE UNDERLYING PHILOSOPHY

The BDI framework is based upon a folk-psychological view of reasoning, that is, the way people *think* that they think, as opposed to the actual mechanics of the way that the brain works. While folk psychology suffers criticism because of the necessity for introspection (as opposed to an objective understanding), it does provide a robust mechanism for reasoning about human reasoning. Bratman's *Intention, Plans and Practical Reason* [1] best summarises the philosophy of the BDI framework (and indeed has been the basis for much work in this field), but an abbreviated summary is given here.

Briefly, an agent is characterised by its beliefs, goals (desires), and intentions – it will *intend to do* what it believes will *achieve its goals* given its *beliefs about the world*. As well as these three components, a BDI agent is usually assumed to have a *plan library* – a set of "plans as recipes" that it can use to achieve particular goals given particular preconditions. An intention is formed when the agent commits to a particular plan – a particular sequence of steps to perform – from this set in order to achieve a goal. The steps themselves may be atomic actions, or they may be subgoals, which can be satisfied by other plans. Because the agent does not need to commit to a particular plan for these subgoals until the last possible moment, this allows a balance between reactive and deliberative planning.

Tying together the beliefs, goals, plan library and intentions is the reasoning engine, as shown in Fig. 1. This reasoner is what drives the agent, updating beliefs, monitoring and updating goals and intentions, selecting plans to achieve goals, and based on the current intentions, selecting the actions to perform.

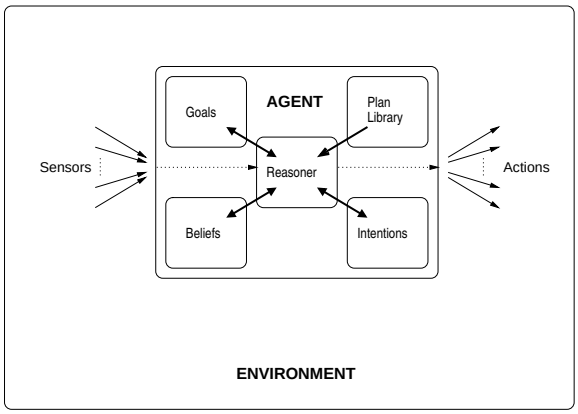

Figure 1: The key components of a BDI agent

A key feature of the plan library is that although the plans are fixed "recipes" for action, they do not have to be fully specified. For any particular goal, there may be multiple plans to achieve that goal, and while any plan *may* be fully specified as a sequence of actions, a plan may instead consist of a sequence of subgoals, or a combination of actions and subgoals. In the case that the plan contains subgoals, the agent can delay the choice of how to achieve a particular subgoal until the time that it reaches that stage of the plan. While this does not achieve the full range of adaptability that people display, it does allow considerable flexibility in the agent's planning, and its resulting behaviour.

In theory, to build a BDI agent, one should be able to specify the things an agent can have beliefs about, the goals that it may wish to achieve, and a set of plans for achieving those goals. Once these three sets have been specified, the agent should be able to operate within its environment, driven by the reasoning engine. Of these three sets, it is in specifying the plans that the bulk of the work lies, but choosing appropriate representations for beliefs and goals is critical. The reasoning in the plans revolves around the agent's beliefs about the world, and so having those beliefs expressed in a suitable form is essential to the development of these plans. Each plan must specify the conditions in which it is applicable, the goal that it aims to achieve, and the steps that need to be performed for success. These steps may be atomic actions that the agent will perform, or they may be subgoals that the agent will attempt to find a plan for when needed.

It is precisely because this philosophy has its foundations in folk psychology that it proves to be useful in capturing human knowledge. When asked about how they think about a problem, people already have a tendency to explain their actions in terms of what their intentions were, which in turn are explained in terms of their goals and beliefs. Moreover, when they describe the ways in which they try to achieve goals (that is, the plans that they use), they will do this in a hierarchical manner, which maps to the partial plans needed for the plan library. Extracting this information from people does require careful planning and structured questioning, but the fact that model builder and the subject being modelled are referring to the same concepts does simplify matters.

3. KNOWLEDGE ELICITATION FOR BDI AGENTS

One of the challenges when building synthetic characters is the task of gathering the knowledge that the characters will need to operate in their world. The complexity of a synthetic character depends on two main factors: the degree of complexity of the environment in which the character is situated (and the degree to which it must interact and respond to that environment), and the complexity of the interactions in which it will be involved, particularly those that involve real people (rather than other synthetic characters, which tend to be more predictable than people). Modern computer games and interactive story-telling environments, like military simulation environments, include characters that tend to the higher end of both of these measures, and it is in this area that the BDI paradigm would be used. Such characters will usually be able to perform a wide range of actions, and combine them in numerous ways to produce an even wider range of behaviours. The challenge is to ensure that the character has a response for whatever situation arises, and moreover that the response in each situation is such that the character behaves in a "sensible" (for that character) manner.

In this project, a form of knowledge elicitation known as

Table 1: Sample First Interview Probes

When you play the game, do you perceive any distinct phases?
What are you main goals in each of these phases?
What are the relative priorities of these main goals?
Say you enter a new game, where you don't know the world map, and you may know some, but not all, of the players. What are the first things that you do?
Do you make an effort to get to know the style of the other players? How do you use that knowledge?

Applied Cognitive Task Analysis (ACTA) was adapted for this purpose. ACTA was originally developed as a means of gathering task knowledge for the development of training programs and for task redesign [9]. It particularly focuses on the cognitive aspects of the task – the reasoning about the courses of action to take, particularly in non-standard cases – rather than trying to define the task in terms of the procedures that must be followed. The methodology consists of a series of semi-structured interviews, during which the subjects are probed about their task, with emphasis on how they use their expertise. The data from these interviews is then used to develop a picture of the *whys* behind the task, in order to develop new procedures and/or interfaces for the task, or to develop training to bring novices quickly up to speed. For the project described here, the same type of semi-structured interviews were used, but the subjects were expert players of Quake 2, and the data was used to build models of these players.

Three different players were involved, each with a quite different playing style, leading to three different models. In this paper, we focus on two of the players in particular: the first was a sniper type player, who would find places to hide and wait for victims to come in to range, and the second was a more aggressive player, who would run around actively searching for victims. In terms of playing skill, the sniper was perhaps slightly the better, but their different styles did not lead to significant differences in their scores.

Each player was interviewed independently, through a series of interviews. The interviewer was *not* an expert at the game, which was an advantage, as it meant that the interviewer did not bring preconceived notions about how to play. The first interview was a "big picture" analysis, consisting of questions such as those in Tab. 1. – did the player perceive the game in terms of different stages; what were the main goals in each stage; what were the relative priorities of the main goals; and so on. The data from this stage was then analysed and used to drive the questions in the subsequent interview. This second interview then probed more deeply about various aspects of the game. In the first interview, the focus was on the player's goals; the focus in the second interview is more about the way the player perceives the world, and probing about special cases. Some examples of the questions that were posed in this stage of the interviews are given in Tab. 2. For two out of the three players a third interview was also necessary. The data from the stage two interview was examined, and there were still gaps in the knowledge. ACTA does not prescribe a fixed number of interviews; they continue until the knowledge gathered appears to be complete. This requires careful analysis of the data at each stage: were there any cases not covered? and were all the terms used fully explained?

Table 2: Sample Probes from Later Stages

You say the first stage of the game is when you don't know the map. When do you consider that you do know the map? Do you explore every nook and cranny?
What makes a good sniping spot?
If you'd just respawned and you could hear but not see a fight nearby, what would you do?
How important are the sounds in the game to you? What sorts of things do you listen for?
What sort of things most clearly differentiate novice players from expert players?
Say you'd identified a particular opponent as being a better player than you. Would you make an attempt to actively avoid him/her?

The subjects in this case were all experienced players of the game, and so the interview questions were designed to extract their expertise. Nevertheless, hypothetical situations were presented in many of the questions, probing to see how the individual would react. An optional stage in the ACTA process is a simulation interview, where an extended scenario is presented to the subject and they are asked to explain what they would do. This stage was not undertaken with any of the subjects, in part because the nature of the game makes it difficult to present such an extended scenario. Another optional stage is to observe the subjects performing their task and then probe them about particular instances that arose. In this project, the subjects participated in an hour-long eight-player game after the final interviews, however no additional questions arose from this exercise.

If the subject was a role-player for a character under development, rather than a player of an existing game, the emphasis of the questions would have to shift but the overall approach would remain the same. There would be a greater tendency towards hypothetical scenarios, and the interviewer would have to be clear in their mind what was needed to achieve coverage. A key outcome of the interviews with the Quake 2 players was that the different players perceived the world in considerably different ways. For example, while all players might refer to "low health," the actual numbers that they gave for it varied significantly. This meant that although the resulting belief structures contained a field that was labelled the same way, the meaning varied from one model to another. In other cases there were things in the world that held significance for one player but not another, and so there were fields in the belief structure of one model with nothing corresponding in another model. The consequence of this for the role-playing case is that the beliefs of the character are largely determined by the role, and capturing the correct "view" of the world will largely influence whether or not the character maintains the suspension of disbelief in the game.

3.1 The Game of Quake 2

Before looking at some examples of how the models of the players were implemented, we first briefly discuss the game of Quake 2 and previous attempts to create automated players, or bots, for the game. It is a well-known first-person shooter game from ID Software which can be played in a number of different modes, both single- and multi-player. For the purposes of this work, it has been the deathmatch version that has been used, which is a multi-player game,

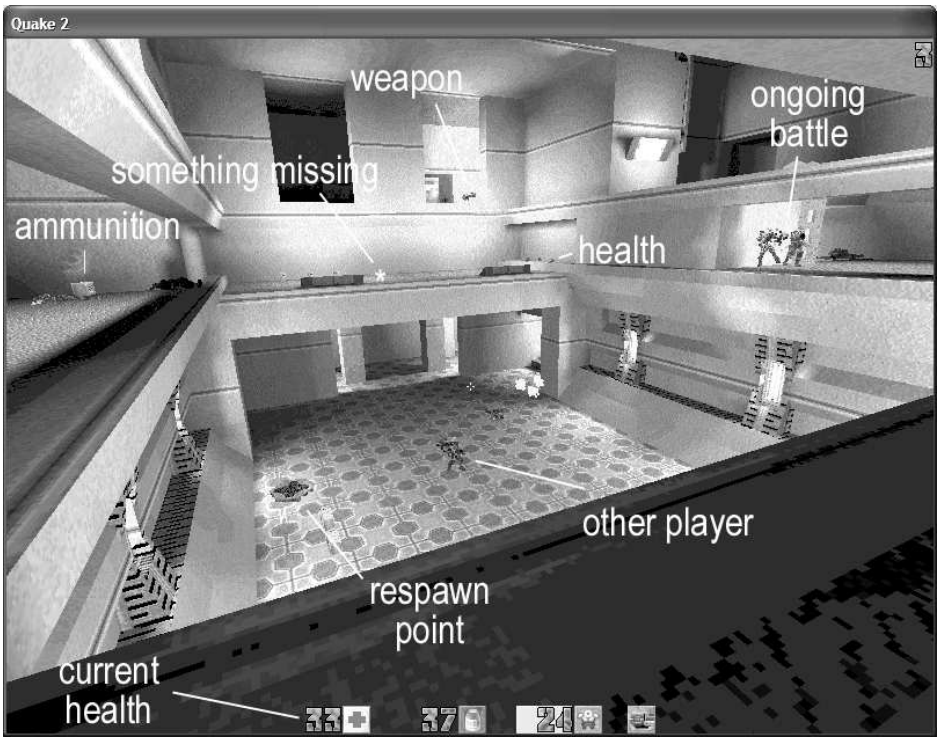

Figure 2: A sample of the detail in Quake 2

the aim of which is simply to get the highest number of kills. The number of players starts at two, and is limited only by the size of the map, but is most commonly in the range of four to ten players in any given game – although this number may include bots as well as human players.

The world of this game is specified by a three-dimensional map, through which the human players navigate with a graphical first person view, using mouse and keyboard inputs. The map can vary enormously from one game to another, from narrow hallways to wide open spaces, and can include hazards, secret hiding spots, bright lights, dark corners, etc. Scattered throughout the map are various items – weapons, armour, health, and others – each of which is periodically regenerated in the same place in the map, and each of which has its own particular set of characteristics. For example, a shotgun has very different range, firing speed, ammunition, etc to a rocket launcher. Players respawn at a number of fixed positions in this map, reappearing at a random one of these each time they die. Each time they are respawned, they have only the most basic weapon, full health, and no other items.

A sample of a player's view is given in Fig. 2. It illustrates some of the detail in the game, although it is somewhat limited by the greyscale format. An expert player would get far more information from this view than is indicated here – such as what type of weapons the other players were using, as well as noticing when they ran out of ammunition. There is also implicit knowledge in the picture: for example, at the position marked * there would usually be a certain gun. The player may know, or at least suspect, that the player who is now on the lower level has just picked this up.

Within this world, the players must equip themselves, seek out other players, and kill them, all while avoiding being killed. (Being killed does not directly affect your score, but puts you at a distinct disadvantage due to lack of equipment.) There are a wide range of strategies used by different players, with none appearing to be significantly better than any of the others. That is, players with completely different styles can be very closely matched in terms of score. Some of the details of how the experts in this study operated are discussed in Sect. 4, but first, consider the differences between human players and existing bots.

3.1.1 *Existing computer generated characters*

There are numerous bots available for download that can play Quake 2 deathmatch with varying levels of success. The most simple of these are obviously "stupid," and experienced players can easily predict their behaviour after watching them for a short amount of time. There are also some very sophisticated bots, which are not so easy to predict, but which nevertheless have obvious flaws in their strategies that experienced human players would not display. As well as these strategic differences that the bots display, they also have low-level differences. The bots have access to raw data about positions, velocities, etc, and this means that they have a tendency to have much better aim than human players, who have to deal with hand-eye coordination.

An example of the type of strategic flaw that bots display is as follows: If two players are involved in a fight, this is an ideal chance for a third player to attack them both. The first two players will already have taken some damage from each other, while the third player is likely to be at full health. Also, the two fighting players are likely to be focused on each other, and so the third player will have the advantage of surprise. In this circumstance, the usual response for both of the first two players, assuming they are human, is to get

away from the fight as soon as they notice the third player. However even the best bots do not react to this situation, continuing to attack their original opponent and ignoring the third player. This is just one example, of which expert players can list many, and which the human players use to their advantage.

4. DESIGN TO IMPLEMENTATION

We now look at some implementations of the strategies of the two players of interest – the sniper and the aggressive player – to demonstrate how the BDI paradigm facilitates the capture of the strategic thinking of the players. The code fragments that are shown are written using JACK, a BDI-based agent programming language, and some brief explanations of the technical details are presented. It should be noted that there are some gaps between the underlying theory of BDI and its implementation in JACK, and that like BDI itself, this language was not designed specifically for human modelling – indeed, it has a wide range of applications. Nevertheless, as the examples illustrate, it is a powerful tool for creating human-like synthetic characters.

Naturally the ultimate goal in the game is to get the highest score – a goal that any player competing in the game will hold. The question is, how do they achieve this goal? At the highest level, both players agreed on the plan to achieve this goal: explore the map, and attack other players. This abstract plan is simply written in JACK:

```
plan Win extends Plan {
    #handles event WinGoalEvent ev;
    #posts event MapGoalEvent map_goal;
    #posts event ScoreGoalEvent score_goal;

    body() {
        @post(map_goal.explore());
        @post(score_goal.attack());
    }
}
```

4.1 Events Instead of Goals

In this very first plan, one of the discrepancies between theory and practice becomes apparent: JACK has no explicit representation of goals. Instead, it has events, which are similar, but not identical. Every plan must handle one and exactly one type of event, equivalent to handling one and only one goal. It may also post events – in this case it posts two – equivalent to setting new goals. The difference between goals and events is that events are instantaneous, and cease to exist after a plan has been chosen to handle them. (Actually, they remain in some sense, because the agent can detect *plan failure* and when this occurs, it can re-post the goal. However there is no explicit representation of the goal that the agent can reason about.)

This means that it is not easy to reason about the goal once a plan has been selected. If, for example, an opportunity arises for a better way of achieving a goal, there is no obvious way of detecting this. Similarly, it is difficult to detect whether a new goal, or a particular plan for achieving a goal, will cause conflicts with existing goals. It is possible to work around these problems, usually by maintaining beliefs with respect to the current goals, but this needs to be done explicitly.

4.2 Sensing the Environment

It was when the subgoals of the two players were examined that the differences began to appear. The goal to explore the map meant quite different things to the each of the players. The sniper was looking for places to hide, places where other players were likely to run by – on their way to a favourite weapon or to boost their health, for example. The aggressive player was also looking for places to hide, but for a different reason. He was looking for places where people were likely to snipe from, and he was looking both for ways to avoid getting in range of these spots, and also for ways to ambush players there. However, the act of observing things in the environment is not just part of the plan for exploring, but something that the players do continuously. This meant that despite the fact the players were noticing different things, the exploration plan also looked the same for both players:

```
plan ExploreMap extends Plan {
    #handles event MapGoalEvent ev;
    #posts event MoveGoalEvent move_goal;
    #reads data MapData map;
    #reads data SelfData self;

    context() {
        map.hasGaps() && !self.fighting();
    }

    body() {
        Position next = map.getNextUnknown();
        @subtask(move_goal.go(next));
    }
}
```

The `@subtask` statement that has been used in this plan is similar to the `@post` statement that was previously introduced. Like the `@post` statement, it creates a subgoal, but whereas the `@post` subgoal is handled asynchronously, this subgoal is handled synchronously, and if the subgoal cannot be achieved, this plan will fail.

The issue of *finding* or noticing things goes back to an issue that was glossed over in Sect. 2, and indeed is often glossed over in the BDI literature: Just how does the agent sense the environment? Is there a plan which receives information from the sensors and updates beliefs, or do the sensors feed directly into the beliefs? In this particular application, the agent connects to a Quake 2 server via an interface. This interface receives a continuous stream of data from the server, which it then decodes and stores as beliefs. The exact beliefs that are stored depend on the player being modelled, and in fact are dependent on the information that the player uses in his plans. Also note that in some cases, changes to beliefs can result in new goals being generated. For example, noticing incoming fire may trigger a goal of "evade attacker."

4.3 Capturing Strategic Differences

The examples given thus far have not shown any difference in the behaviour of the two players. However the plans used to achieve the second goal of the top level plan, to build the score, did show clear differences:

```
plan BuildScore extends Plan {
    #handles event AttackGoalEvent ev;
    #posts event EquipGoalEvent equip_goal;
    #posts event HideGoalEvent hide_goal;
    #uses data SelfData self;
    #uses data MapData map;

    context() {
        !self.fighting() && !map.hasGaps()
            && !self.seePlayer();
    }

    body() {
        @subtask(equip_goal.getWeapon());
        @subtask(hide_goal.goSnipePos());
        @wait_for(self.seePlayer(), TIMEOUT);
    }
}
```

This plan is the plan used by the sniper – in the case that he has explored the map, is not already engaged in a fight, and cannot see another player. Basically, he makes sure he has a "decent" weapon (which is handled as a subgoal), then selects a suitable hiding spot, goes there and waits for a player to appear. If by some chance a player does not appear within a given period, the plan fails. If such failure occurs, the agent will re-plan, which in this particular case is likely to involve this plan again, but a different hiding spot. If the plan succeeds, the agent will have seen another player, and this will itself generate another goal, via the server interface belief discussed above. Contrast this with the plan body of the more aggressive player in the same context:

```
    body() {
        @subtask(equip_goal.getWeapon());
        while(!self.seePlayer()) {
            Position pos;
            if ((pos = self.hearPlayer())
                    == null) {
                pos = map.getLikelySpot();
            }
            @subtask(move_goal.go(pos);
        }
    }
```

Again, this agent wants to have a "decent" weapon before hunting out other players. The difference is, once he has one, he will move from place to place, actively seeking out players. This puts him at greater risk, since the other agent is not as visible to other players, but it also means that he is more likely to encounter other players, particularly in a sparsely populated map.

4.4 The Case of the Third Combatant

These two plans demonstrate the ease with which the strategic differences of different players can be captured, but what about the motivating example, where humans will retreat from a fight but bots won't? This is handled using a maintenance condition, which means that the agent will only continue to fight a given player while it is "safe" to do so:

```
plan HandlePlayer extends Plan {
    #handles event PlayerSeenEvent ev;
    #posts event EngageGoalEvent engage_goal;
    #posts event EvadeGoalEvent evade_goal;
    #uses data SelfData self;

    context() {
        !self.fighting();
    }

    body() {
        if (self.shouldAttack(ev.player))
            @maintain(self.safeFight(),
                engage_goal.attack(ev.player));
        @post(evade_goal.hide());
    }
}
```

The agent checks to see if it should attack the player specified in the goal (based upon the agent's own status and its knowledge of the other player). If it should, it will engage the player, and continue fighting this player while it is "safe." Thus far, all discussion about the implementation has concerned plans, with passing reference to the beliefs and goals (events) within the plans. To demonstrate how the maintenance condition works, a fragment of `SelfData` is presented:

```
public view SelfData {
    #uses data Health health_bel;
    #uses data Fight fight_bel;
    ...

    #complex query safeFight() {
        return health_bel.safe() && ...
            && fight_bel.singleOpponent();
    }

    boolean fighting() {
        return fight_bel.numOpponents() > 0;
    }

    ...
}
```

Every time the `health_bel` or `fight_bel` beliefs change (via new information arriving from the server), the `safeFight()` conditions are re-evaluated. There were other conditions in this test than are shown here, but these are omitted for brevity. `safeFight()` is a *complex query*, whereas `fighting()` is simply a boolean function. This is because `fighting()` is only ever used as a simple test, rather than as a trigger for a maintenance condition (or one of the other types of JACK statements – not illustrated here – that requires a triggering condition).

4.5 Representing Beliefs

In JACK, there are a number of mechanisms for representing beliefs. Most simply, they can be instances of any Java class or basic data type, but there are two special JACK constructs that give extra functionality: views and beliefsets. These types of beliefs can be used to trigger new goals in the agent, a way of achieving reactive behaviour in the agent. The previous example is a view, which is itself a compound belief, referencing other beliefs including `health_bel`

and `fight_bel`. A beliefset is a simpler construct, containing only Java constructs. An example of a changing belief triggering a new goal is illustrated in the `Health` beliefset:

```
public beliefset Health extents OpenWorld {
    #posts event EvadeEvent evade_goal;
    #value field int health;
    #indexed query getHealth(logical int $h);

    #complex query safe() {
        logical int $health;

        return getHealth($health)
            && $health.as_int() > MIN_HEALTH;
    }

    void addfact(Tuple t, BeliefState is) {
        Health__Tuple ht = (Health__Tuple)t;
        if (ht.health < CRIT_HEALTH)
            postEvent(evade_goal.panic());
    }

    ...
}
```

This beliefset has a single field, the value of the health that is received from the server. When this value is updated (by adding a fact to the beliefset), the `addfact` callback is triggered (a built in feature of JACK), and if the health has fallen below a critical level, a new goal is posted. The fields of a beliefset are accessed by logical variables, which is the reason for the somewhat unusual syntax above. These are a feature of JACK, and allow prolog-like queries of beliefsets, which can be a powerful mechanism when writing plans.

4.6 Low Level Skill Differences

In all the code examples above, there are no atomic actions – that is, steps in the plan where the agent actually *does something* in the environment. The steps in the plans are subgoals, which may themselves be achieved by plans consisting entirely of subgoals, but eventually they must be grounded in actions – which in the case of this application means sending simulated keyboard or mouse events to the server.

However once we reach this level, the people being modelled do not actually *think* about these actions. Their reasoning is at the level of "*I move to position A, hugging the walls*," not the details of which keys they press to achieve this. In fact this is done as an action-feedback loop, where the player presses keys (or moves the mouse) based upon the view they see on their screen. Fortunately studies in HCI and user interface design provide objective data on this type of behaviour, including the time taken for mouse movement and key presses, as well as the errors that arise [3], and previous work has already demonstrated how this can be incorporated into a model built in JACK [13]. Thus the low level actions can be implemented independently of the strategic information supplied by the expert players.

This also has an important side effect: due to the fact that this data already includes models of error in movement, and the time taken for actions, the errors that human players make (in terms of aiming and movement) are automatically incorporated into the model.

5. LIMITATIONS AND ONGOING WORK

As discussed in Sect. 2, the BDI paradigm is useful in capturing human knowledge because of its folk psychological foundations. The folk psychological model of reasoning is a way of dealing with the complexity of human reasoning. It is powerful because it gives a robust model of human reasoning without having to explain the mechanics of how the brain works. However there are circumstances where the paradigm is overly complex for what is required, and other circumstances where the level of abstraction is too high, and a more detailed model is appropriate.

The power of a BDI solution lies in the ability to abstract a complex environment in which complex strategies are used. It is when the people being modelled describe their reasoning in folk psychological terms that BDI is useful. When the people can describe their reasoning in less complex form, a solution that matches that form will almost certainly be less computationally complex than a BDI-based solution, and probably more easily implemented. However the range of environments in which people describe their reasoning in folk psychological terms is extensive, and the BDI framework is a powerful tool for building models of these people in these environments.

There are cases though when the level of abstraction provided by the BDI paradigm is too high. These fall into two broad sub-categories: those where low level subconscious behaviours have a strong influence on the overall behaviour of the character, and the second is where there are additional high-level conscious processes (for example memory, emotion or learning) that have a significant influence. Fortunately the BDI framework can be extended to support characteristics that fall into either of these categories.

The first case, of subconscious behaviours, can be implemented as "plug ins" to the framework. These behaviours are not directly part of the reasoning process, but influence it because the character must adjust for the timing and errors that occur in the behaviours. The low level skills described in Sect. 4.6 are one such example of this – the players did not think about the way the used they keyboard and mouse, yet the timing and errors meant that the agents had to adjust their actions and reasoning to take these into account.

In the second case, where the processes of interest are high level conscious processes, the framework must be adjusted to incorporate models of these processes within it. Fortunately the folk psychological roots are again a benefit here. For any such conscious process, there is a folk psychological explanation for it – that is the nature of folk psychology – and such an explanation will generally use similar concepts (and possibly some additional ones) to those used in the BDI paradigm. This facilitates the development of a new framework, based on BDI, that incorporates the characteristic of interest. One example of this is presented by Norling [12], where the decision making model used by the agents is modified, to use recognition-primed decision making (RPD) [6]. Gratch and colleagues have made a similar type of extension to the Soar architecture, this time addressing the issue of emotion [7]. [1]

[1]While Soar is not a BDI architecture in the strict sense, conceptually it is often treated in this way. The model of emotion used by Gratch et al. does use concepts that relate closely to those already used in BDI-based languages.

6. CONCLUSIONS

The BDI paradigm is a powerful tool for building models of human operators performing complex tasks in complex environments. The underlying philosophy of the paradigm maps easily to the way in which people tend to explain their behaviour in these situations, and as such it provides a basis for the design of these agents. In moving from this design to implementation, there will inevitably be some compromises due to the gap between theory and practice, but experience has shown that the effort required for this compromise is more than compensated by the ease of design. The example presented in this paper has illustrated the ease with which high level strategy can be captured in the agents' plans, and also highlighted some of the gaps encountered, showing how they were dealt with.

Gathering and representing the knowledge needed to build a synthetic character is always a challenge, but the modified ACTA methodology described in Sect. 3 has been highly successful. ACTA has been developed to understand the reasoning behind the performance of tasks, in much the same way that the BDI paradigm captures the reasoning that leads to action. The interview process allows the builder to determine the appropriate representation for the beliefs, goals and plans of the character under development. This appropriate representation greatly facilitates maintaining the integrity of the character's role, and subsequently the of suspension of disbelief that is so important in synthetic environments.

Some examples have been presented of the capture of high level strategies of Quake 2 players using the JACK agent language. These illustrate that while the language was not designed specifically for human modelling, the BDI basis of the language does allow these strategies to be relatively clearly and succinctly expressed. Some adaptations had to be made to account for gaps between the theory and practical implementation of the paradigm, but these were relatively easy to achieve. Computational resources would of course limit the number of characters of this complexity that could be supported in a game, but with appropriate tailoring of the system, at least key characters could be implemented using the BDI paradigm, as in *Black&White*, and using level of detail switching techniques, it may be possible to support a larger number of characters in this way.

Finally, while the BDI paradigm is not the solution for all synthetic characters, it is extremely powerful for those that must exhibit a wide range of complex behaviours, particularly when they must interact with (often unpredictable) players. In some cases it does not capture the all the aspects of human behaviour that are of interest, but even in these cases, it provides a basis for an extended framework. Subconscious behaviours can be attached to the existing framework, so that they affect behaviour directly but reasoning indirectly, and conscious processes can be integrated into the framework, using folk psychological concepts compatible with those already used.

7. REFERENCES

[1] Michael E. Bratman. *Intention, Plans and Practical Reason*. Harvard University Press, Cambridge, Massachusets, 1987.

[2] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[3] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[4] Clint Heinze, Bradley Smith, and Martin Cross. Thinking quickly: Agents for modeling air warfare. In *Proceedings of the Eighth Australian Joint Conference on Artificical Intelligence*, Brisbane, Australia, 1998.

[5] Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. JACK intelligent agents – summary of an agent infrastructure. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Montreal, Canada, May 2001.

[6] Gary A. Klein. A recognition-primed decision (RPD) model of rapid decision making. In Gary A. Klein, Judith Orasanu, Roberta Calderwood, and Caroline E. Zsambok, editors, *Decision Making in Action: Models and Methods*, pages 138–147. Ablex Publishing Corporation, 1993.

[7] Stacy Marsella and Jonathon Gratch. Modelling coping behavior in virtual humans: Don't worry, be happy. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 313–320, Melbourne, Australia, July 2003. ACM Press.

[8] David McIlroy and Clinton Heinze. Air combat tactics implementation in the Smart Whole AiR Mission Model (SWARMM). In *Proceedings of the First International SimTecT Conference*, Melbourne, Australia, 1996.

[9] Laura G. Militello and Rob J. B. Hutton. Applied Cognitive Task Analysis (ACTA): A practitioner's toolkit for understanding cognitive task demands. *Ergonomics*, 41:1618–1641, 1998.

[10] Peter Molyneux. Postmortem: Lionhead Studios' Black & White. *Game Developer*, June 2001.

[11] Graham Murray, Serena Steuart, Dino Appla, David McIlroy, Clinton Heinze, Martin Cross, Arvind Chandran, Richard Raszka, Gil Tidhar, Anand Rao, Andrew Pegler, David Morley, and Paolo Busetta. The challenge of whole air mission modelling. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, Melbourne, Australia, 1995.

[12] Emma Norling. Learning to notice: Adaptive models of human operators. In *Second International Workshop on Learning Agents*, Montreal, Canada, May 2001. ACM.

[13] Emma Norling and Frank E. Ritter. Embodying the JACK agent architecture. In Markus Stumptner, Dan Corbett, and Mike Brooks, editors, *AI 2001: Advances in Artificial Intelligence*, volume 2256 of *Springer Lecture Notes in Artificial Intelligence*, pages 368–377. Springer, 2001.

[14] Adrian Pearce, Clinton Heinze, and Simon Goss. Enabling perception for plan recognition in multi-agent air-mission simulations. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS2000)*, pages 427–8, Boston, July 2000. IEEE Computer Society.