Spyke3D: a new Computer Games oriented BDI Agent Framework

Luca Palazzo, Gianluca Dolcini, Andrea Claudi, Gianluigi Biancucci
Paolo Sernani, Luca Ippoliti, Lorenzo Salladini, and Aldo Franco Dragoni

Dipartimento di Ingegneria dell'Informazione (DII), Università Politecnica delle Marche, Italy
{l.palazzo, a.f.dragoni, g.dolcini, a.claudi, p.sernani}@univpm.it

***Abstract*—One of the video game industry main goal is to meet the players' ever growing yearning to experience more immersive and realistic virtual worlds. Modern games usually answer to this demand with advanced computer graphics and scripting techniques, rather than opting for "strong" Artificial Intelligence features, in spite of the scientific community several calls and proposals.**

In this paper we introduce Spyke3D, a Multi-Agent Framework based on Belief-Desire-Intention paradigm, whose purpose is to support and simplify the developing of human-like logical structures, in order to bestow more realistic and plausible behaviours on Non-Player-Characters of the world, providing a further convincing feeling to the game.

***Index Terms*—Computer Games, Multi-Agent Systems (MAS), Beliefs-Desire-Intention (BDI), Expert Systems, Non-Player-Characters (NPC).**

I. Introduction

The Game Industry has invested heavily, over the years, to make computer games ever more technically sophisticated. A high photorealism level, however, doesn't necessarily mean a lofty degree of immersion and involvement with the virtual world. There is a growing demand, especially by seasoned players, for more plausible NPCs' behaviours to keep up with the realism of models and environments [1,2]: one of the most craved wish is to bring the offline gaming experience closer to the online one, where, generally, both allies and enemies are played by human gamers.

AI algorithms have always placed side by side with computer games development, but rarely they can be considered as belonging to the strong AI concept, as their purpose is to perform specific tasks (e.g. state space exploration), rather than reproducing human cognitive processes. There are, however, several games in which experience itself could be more essential than victory or defeat, where the player's avatar constantly needs, to reach his goals, to relate with the world NPCs: the application of strong AI features - such as a logical description of mental attitudes, decision-making and inference processes, etc. - could way better express this kind of interactions.

Laird and van Lent [3] made a meticulous AI classification by computer games genres and, for each one, they discriminated NPCs depending on their role in the game: through this survey they locate where the adoption of a strong AI may considerably improve the gaming experience. Dignum [4], in particular, affirms that such approach is especially suited for serious computer games, which, more than others, require very natural and convincing behaviours for NPCs.

Related Works

During the workshop series on "Agent for Games and Simulation" [5,6], started in 2009, various communication modes between agents and game engines have been argued, and most surfaced issues derive from the different nature of their design: game engines provide a virtual world representation based on a centralized control of the game; agents are inherently autonomous and proactive entities, whose interaction with other agents or with the environment (the game engine itself in this case) is asynchronous. For this reason Dignum et al. [7] make a distinction between the following approaches:

- *Server-side*. It's the standard AI developing method for computer games: the agent's decision-making process is entirely managed by the game engine and must be performed within a default time slot.
- *Client-side*. Agents are considered as separated applications of a client, which communicates with the game engine through asynchronous messages: Gamebots [8], Flexbot [9], Quakebot [10] and Pogamut [11], all mainly born for research reasons, are shining examples of this approach. A further game design effort is also needed and it often represents a serious constraint: according to Rabin [12] and Schwab [13], AI is usually added at the end of the game development and, at this level, the introduction of agent oriented programming may even bring to worse results.

Following the latter approach, Dignum et al. [7] made a vast survey about the steps and issues to address in order to adopt MAS technology in computer games. Regarding the BDI paradigm [14], an agent is described, through logical statements and rules, by its own mental attitudes, such as desires and beliefs about its surrounding environment. Agents are also provided with a sort of human-like practical reasoning: a deliberation system makes agents undertake intentions in order to activate consistent plans (set of actions to run within the environment). According to Dignum et al. [7], an agent should not expose this reasoning process to the game engine, but just those significant actions for the render scene: decision-making phase is performed at a logical-symbolic level, while the environment is managed at a geometrical one; message exchange between the two levels is needed in order to achieve an updated and consistent knowledge base, and to show results of chosen actions within the world.

During the years, there have been certain attempts to introduce BDI based NPCs in computer games. Davies and Mehdi [8], Patel and Hexmoor [15], Weber et al. [16] modeled BDI, or conceptually similar paradigm, bots for respectively Unreal Tournament [17], Counter-Strike [18] and StarCraft [19]: in each case BDI agents made bots more believable, providing a generic more realistic feeling to the game. However, some restrictions arise from this approach: BDI paradigm itself is not able to fully reproduce human-like cognitive processes [4]; moreover, as stated by Davies and Mehdi [8], adding a different bot concept on an existing game at a later stage is not an ideal solution. This process must, instead, occur concurrently with the game development, in order to better evaluate the agent behaviours.

Paper Contribution

The aim of this paper is to introduce Spyke3D, a MAS and expert system technology-based framework, whose purpose is to support the modeling of BDI agents for a virtual 3D environment, in order to facilitate the use of such technologies in the field of computer games development. Our proposal is born by taking care of guidelines and issues exposed in the previous section and comes up as a framework: one of its main features is to exhibit to game developers a unique environment for a simplified agent management, hiding the reasoning mechanics and other technical details.

The name Spyke3D originates, with not so much imagination effort, from a sort of crasis of the used applications names: SPADE [20], a FIPA compliant multi-agent platform; PyKE [21], a knowledge base inference engine; and Panda3D [22], a free and open source 3D game engine. A common feature of all these programs is Python programming language: this choice is due to Python core philosophy of a faster high-level development and a more readable code; moreover there has been, during the years, an increasing opening to this programming language by the game developers community [23].

II. Spyke3D Framework

This section presents Spyke3D and its features. In order to better explain the capabilities of the framework, we start from the description of tools and libraries we use, and how they have been useful in order to achieve our goals. An accurate description of implemented features will follow, while this chapter will end with the introduction of a test scenario in order to show the framework working principles.

A. Working Tools

1) SPADE: SPADE [20] is a Multi-Agent Platform written in Python, which offers many features and facilities to simplify the construction of a MAS. Among its qualities we can enumerate (Fig. 1):

- SPADE is a IEEE FIPA [24] compliant platform, so it can be used to build a modular system, where agents are even able to interact with other agents written in different programming languages and/or belonging to other FIPA compliant platforms. The message exchange is defined by FIPA-ACL protocol, while the content languages supported are both FIPA-SL and RDF: FIPA ACL messages are embedded in one of the Message Transport Protocol (MTP) supported by SPADE, for example XMPP.
- SPADE implements four MTPs: XMPP (default), P2P, HTTP and SIMBA. SPADE is the first agent platform to base its roots on XMPP [25]: an open, extensible, asynchronous, distributed and secure XML based communication protocol.
- SPADE provides a Python module called Agent Library to ease the agent creation and interaction within the platform. SPADE developers are also considering the opportunity to issue APIs for different programming languages, C# and Java in particular. SPADE Agent Library also provides a support for BDI paradigm, but it is still in an embryonic stage for the goals of this work: it briefly consists in a wrapper of a first-order logic python library developed by Peter Norvig to solve exercises from his well-known book AI:MA [26]. For this reason, we decided to implement a new management system for logical structures and reasoning processes of SPADE agents, grounded on the adoption of an expert system library: PyKE.

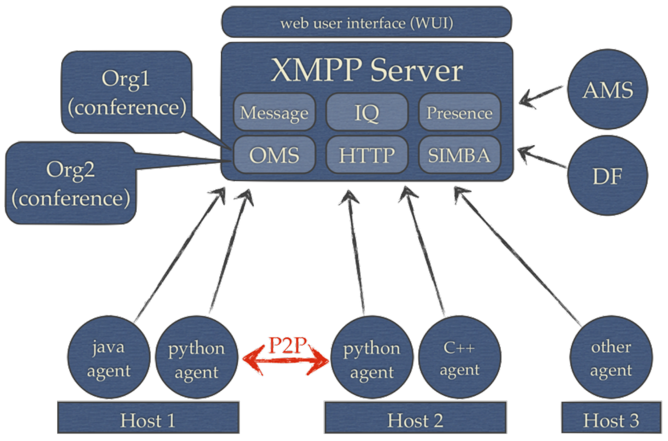


Figure 1: SPADE platform overview.

2) PyKE: PyKE [21] is an open source library which makes use of a Prolog inspired logic programming language, in order to provide a knowledge-based inference engine to the Python community. PyKE's engine is based on two concepts, denominated facts and rules: the former stand for the symbolic knowledge that the expert system has about the world; the latter are needed to run data (facts) inferences and to bind goals with plans. More in detail:

- *Forward Chaining Rules* are processed automatically if the corresponding rule set is activated. A forward chaining rule "fires" when its premises are true, and the achieved effect is the assertion of new statements, which will increase the list of known facts about the world. Following this concept, we extended PyKE's architecture introducing what we called *Retract Chaining Rules*: contrary to what happens with forward chaining rules, they

allow to retract specific known facts if some premises are verified. In this way, we could apply belief revision techniques, satisfying AGM postulates [27], to maintain a consistent knowledge for the learning agent.
- *Backward Chaining Rules* are activated asynchronously when a question is asked to the inference engine, in a Prolog analogous way. PyKE will search for a rule whose conclusion field contains the requested goal, and whose body is composed of verified subgoals or known facts of the world: if so, the asked question is verified. PyKE allows to assign plans to backward chaining rules, so that we can define a set of actions which the agent has to carry out in order to reach a desired goal.

PyKE allows developers to define facts and rules in terms of logical programming statements and clauses, then, for performance purposes, it compiles them in order to generate new Python modules to call inside applications.

3) Panda3D: Panda3D [22] is an open source game engine made by Disney in collaboration with Carnegie Mellon's Entertainment Technology Center. It is written in C++ with a set of Python bindings for a faster game development and it comes with a full set of libraries to manage audio, collisions, input and physics, in addition, of course, to real time rendering. We use Panda3D merely for NPC's "body" modeling, because NPC's reasoning side is completely unrelated with the game engine.

B. Framework Features

This section introduces Spyke3D's architecture, in terms of agent and knowledge management, describing how it can support and simplify the development of BDI based NPCs in a 3D environment.

1) NPC's Structure: Following Dignum et al. [7] directives we decided to partially decouple NPCs from the game engine. In detail, any Spyke3D NPC is an entity composed of two interacting agents representing the NPC's "brain" and "body" (Fig. 2): the former is responsible for reasoning, while the latter is a game engine wrapper. They are provided with a dedicated communication channel in order to reproduce a sort of peripheral nervous system: sensory data are exchanged from body to brain, while intentions (actions to undertake) follow the opposite way. Advantages from this approach are:

- *AI distribution*: this solution allows to distribute AI computations to unload machines which own rendering tasks.
- *AI independent from programming language and game engine*: the interaction between brain and body agent is FIPA based, so that AI can be developed ignoring game engine specifications and, consequently, body agent implementation details; it is sufficient that brain and body agent share the same communication interface. Another interesting benefit from a such approach is an AI code complete reuse for further works.

2) Agent DNA: Human behaviours are influenced by both environmental and genetic factors. In this manner, if an agent should learn from experience, it can also inherit some traits during its creation. More in detail, brain agents have a set

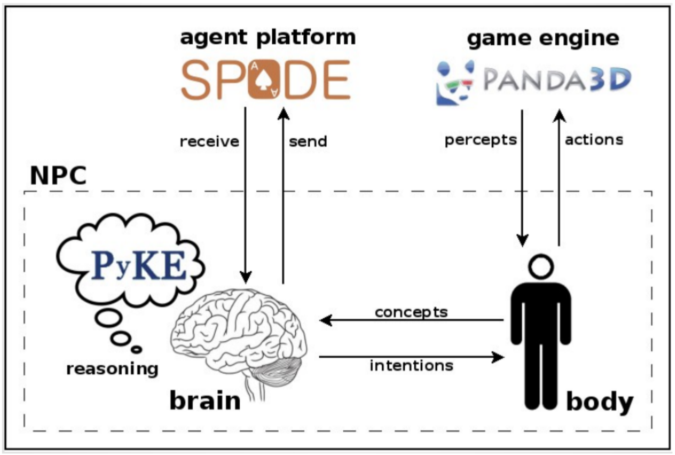


Figure 2: NPCs are detached in brain and body agents.

of attributes which may influence the deliberation phase in terms of goal and plan scheduling. In this way, even NPCs which share the same knowledge base could behave differently because of their "genes".

3) Intentions: NPC's DNA makes behaviours more or less deterministic, conditioning the agent deliberation phase. It is necessary to consider that human behaviours may even be irrational, insomuch as we are often inclined to pursue certain goals regardless of their optimality. In Spyke3D, goals are Python classes that represent world states which an agent wants to be true, plus a set of metadata as their type (e.g. maintain goal), deadline and desirability value.

- *Goal Deliberation*: during this phase, the agent selects an objective to pursue (if it has one). The agent list of wishing goals is scheduled depending on their metadata and agent DNA: more rational agents will tend to choose a more worthwhile goal, chaotic agents may pick up a goal randomly, while irrational agents could select to pursue a low desirable goal.
- *Plan Deliberation*: once the goal is chosen, the agent must define how to pursue it. It queries its knowledge base in order to pull out, through backward chaining rules, all the plans which are congruent with the agent beliefs and lead to the desired world states. Then, the plan choice is based on attributes like its cost and its satisfaction level (variable weights grounded on the agent experience).

Human beings usually are not able to consciously perform more tasks simultaneously, so agents should follow this conduct too. Agent DNA has an attribute for the maximum number of parallel foreground tasks, after that the agent deliberation phase is suspended.

4) Plans and Behaviours: In agent oriented programming, agent tasks are performed by particular computational models called behaviours. In BDI paradigm, plans are action sequences to undertake in order to pursue a chosen goal. With Spyke3D we bound PyKE's backward chaining rules, which can be linked to Python written plans, with SPADE behaviours, in order to transfer plan actions to the agent platform: to achieve this task, rules must contain in their

```
my_spyke3D_plan
    use my_goal($v1, $v2) taking (agent, goal)
    when
        facts.my_fact($v1)
    with
        from mymodule.mybehaviours import MyBehav
        agent.add_goal(MyBehav(Goal, $v2))
```

Figure 3: Backward chaining rule linked to a plan with an agent behaviour.

namespace the agent variable so that plans can be executed within a specific agent behaviour (Fig. 3), belonging to what we called BDIBehaviour class type.

5) Stereotypes: Individuals moulded in the same environment are inclined to share part of their knowledge: for example soldiers have some common skills, even if an archer has a different role compared to a swordsman. Spyke3D allows an easier creation of fact and rule set through knowledge partitioning. These knowledge modules represent a sort of "domain" knowledge that a NPC could have, and they can be added during the agent creation without requiring code duplication.

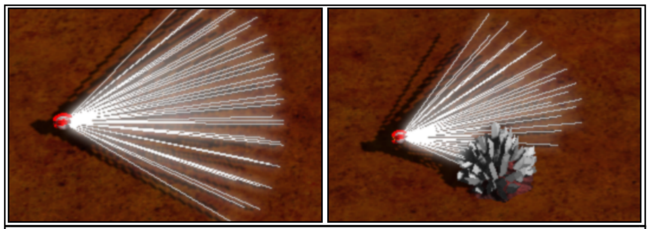
6) Game Engine Interaction: Body agents represent both sensors and actuators of our system: as we stated, they have the role to interface with the game engine. For this reason, body agents can not be defined by developers a priori, but on a case by case basis. However, Spyke3D provides a communication interface, which makes use of Python dictionaries and a specific "b2b" (body to brain or vice versa) ontology, to simplify the interaction between NPC's brain and body agents.

C. Test Scenario

In this section we'll see Spyke3D at work. The ideal test-bed for our framework is clearly represented by a real game development, so that we could evaluate both AI design and implementation phases, and game enhancements in terms of its credibility, unpredictability and realism degree. Unfortunately, building a killer game application is beyond our possibilities, so we limit this test to the realization of a basic scenario whose main goal is to show Spyke3D's working principles, focusing on the interaction with the game engine and communication between brain and body agents.

1) Description: We modeled two kinds of NPCs: explorers and gatherers, each one with his skills, knowledge and goals. We put them in a 3D environment, unknown to them, containing different kind of resources. Both explorers and gatherers have the final goal to carry more valuable resources to their home base, but their interaction is needed to perform a such task, because only gatherers can bring resources, whereas explorers can identify them and are endowed with a keener eye.

2) Implementation: 3D models and animations were created with Blender [28] and exported to a Panda3D readable format (.egg) using Yabee [29]. Through body agents, we modeled NPCs' animations and senses, such as sight and hearing, in order to allow their interaction within the environment. As we stated, body agents interface directly with the game engine, in fact senses are implemented using Panda3D collision system (Fig. 4): a cone of collision segments for sight and a collision sphere for hearing, both affected by material depending occlusions through the adoption of different collision bitmasks.



```
def collisionView(self):
    self.view_np = self.lego.attachNewNode(
      CollisionNode('view_np_'+self.name))
    self.view_np.setPythonTag('owner', self)
    self.view_np.node().setIntoCollideMask(BitMask32
      .allOff())
    self.view_np.node().setFromCollideMask(BitMask32
      .bit(1))

    for i in range(0,31):
        d = view_solids[i]
        x = -30 + i*2
        y = -math.sqrt((math.pow(d, 2)-math.pow(x,
          2)))
        segment = CollisionSegment(0,0,3, x, y, 0.1)
        self.view_np.node().addSolid(segment)
    base.cTrav.addCollider(self.view_np, base.cHan)
```

Figure 4: Body agent sight using Panda3D collision system.

Once a new sensory event is fired (e.g. a new resource object is found) body agents will send such information to their relative brain by way of FIPA-ACL messages, causing its knowledge base update. In a similar way, NPC's brain deliberation system produces intentions - such as wandering, gathering, moving to, communicate with, etc. - to forward to their body agent, which perform their wrapping with the environment, translating actions to game engine interpretable instructions. In order to ease body/brain agent communication, Spyke3D encapsulate *_handle_message()* method call in a SPADE cyclic behaviour, so that it is just necessary to override it and define a dictionary of possible message types (Fig. 5).

```
import spyke

class ScoutBrain(spyke.agent.Brain):
    """ Scout's brain """

    def _handle_message(self, message):
        """ Handles messages from body """
        msg_type = message.get('rdf:type')
        if msg_type == "view":
            what = message.get('rdf:what')
            (x,y,z) = eval(message.get("rdf:where"))
            self.add_fact("position", (what, x, y ,z
              ))
        ...
```

Figure 5: Section of a brain agent message handling.

3) Results: From the scenario building it emerges how Spyke3D can be useful to model, without much effort, BDI based NPCs for a gaming-oriented application. Developers, in fact, do not need to care for how decision-making process occurs or how NPCs' body/brain communication is managed, but they must implement behaviours extending specific classes, build a knowledge base using PyKE's logical programming language and handle agent messages. Naturally, a superior effort is needed for body agent implementation, which must reproduce brain deliberated intentions within the environment.

One of the framework strong point is its capability to abstract brain reasoning process from game engine constraints, therefore developers could reuse NPCs' brain agent implementation for further applications, even with different contexts, as if it was a software library.

III. Conclusion

This paper confirms results from several research works affirming that agent oriented programming could be a very interesting approach for computer games AI development. BDI paradigm, in particular, directly representing logical mental attitudes and their dynamics, allows to reproduce human-like cognitive processes which can seriously enhance NPCs' behaviours in terms of realism, credibility and naturalness. If these aspects are important in playful contexts, they are essential when learning and training are the application purposes, as it happens for serious computer games.

Spyke3D, despite its earlier state, intends to offer a tool for a faster entrance of such technologies in the computer games sphere. In this paper we described its features and how they can be combined with the game development process. However, several tests are still needed to better highlight the framework capabilities: a complete game AI development would be, in fact, the ideal test-bed to evaluate Spyke3D features.

Future Works

Spyke3D is still a work in progress: many features can be implemented to extend framework capabilities.

- *Fuzzy Logic*. We are adding to PyKE a fuzzy control system to increase knowledge coarseness in order to better manage uncertainty states.
- *GUI*. A GUI presence could further help developers to create BDI base NPCs using Spyke3D.
- *Deliberation*. Goal and plan scheduling system is still naive: the implementation of a more advanced algorithm able to manage goal starvation could be opportune.
- *Body Agents*. Body agents should be implemented for more game engines, so that we could evaluate AI development reuse in different contexts.

References

[1] Johnson, Daniel, and Janet Wiles. "Computer games with intelligence." Fuzzy Systems, 2001. The 10th IEEE International Conference on. Vol. 3. IEEE, 2001.

[2] van Lent, Michael, and John Laird. "Developing an artificial intelligence engine." Ann Arbor 1001 (1998): 48109-2110.

[3] Laird, John, and Michael van Lent. "Human-level AI's killer application: Interactive computer games." AI magazine 22.2 (2001): 15.

[4] Dignum, Frank. "Agents for games and simulations." Autonomous Agents and Multi-Agent Systems March 2012, Volume 24, Issue 2, pp 217-220.

[5] Dignum, F., Bradshaw, J., Silverman, B., & van Doesburg, W. (Eds.). (2010). "Agents for games and simulations: Trends in techniques, concepts and design." LNAI 5920. Heidelberg: Springer.

[6] Dignum, F. (2011). "Agents for Games and Simulations II: Trends in techniques, concepts and design." LNAI 6525. Heidelberg: Springer

[7] Dignum, Frank, et al. "Games and agents: Designing intelligent gameplay." International Journal of Computer Games Technology 2009 (2009).

[8] Davies, N. P., and Qasim Mehdi. "BDI for intelligent agents in computer games." (2006). The proceedings of CGAMES'2006, ISBN 0-9549016-1-4.

[9] Khoo, Aaron, et al. "Efficient, realistic NPC control systems using behavior-based techniques." AAAI Spring Symposium on Artificial Intelligence and Computer Games. 2002.

[10] Laird, John E., and John C. Duchi. "Creating human-like synthetic characters with multiple skill levels: A case study using the soar quakebot." Ann Arbor 1001 (2000): 48109-2110.

[11] Gemrot, Jakub, et al. "Pogamut 3 can assist developers in building AI (Not only) for their videogame agents." Agents for Games and Simulations. Springer Berlin Heidelberg, 2009. 1-15.

[12] Rabin, Steve. AI game programming wisdom. Cengage Learning, 2002.

[13] Schwab, Brian. AI game engine programming. Hingham: Charles River Media, 2004.

[14] Wooldridge, Michael, and Nicholas R. Jennings. "Intelligent agents: Theory and practice." Knowledge engineering review 10.2 (1995): 115-152.

[15] Patel, Purvag, and Henry Hexmoor. "Designing BOTs with BDI agents." Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on. IEEE, 2009.

[16] Weber, Ben G., Michael Mateas, and Arnav Jhala. "Building human-level ai for real-time strategy games." Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems. 2011.

[17] Epic Games Inc. Unreal Tournament. http://www.unreal.com/

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

[19] Blizzard Entertainment. StarCraft. http://www.starcraft.com/

[20] SPADE: Smart Python multi-Agent Development Environment. https://github.com/javipalanca/spade/

[21] PyKE: Python Knowledge Engine. http://pyke.sourceforge.net/

[22] Panda3D. Carnegie Mellon's Entertainment Technology Center. http://www.panda3d.org/

[23] McGugan, Will. Beginning Game Development with Python and Pygame. Will McGugan, 2007.

[24] FIPA. The Fondation for Intelligent Physical Agents. http://www.fipa.org/

[25] XMPP. Extensible Messaging and Presence Protocol. http://xmpp.org/

[26] Russell, Stuart Jonathan, et al. Artificial intelligence: a modern approach. Vol. 2. Englewood Cliffs: Prentice hall, 2010.

[27] Katsuno, Hirofumi, and Alberto O. Mendelzon. "On the difference between updating a knowledge base and revising it." (1991).

[28] Blender. Free and open 3D creation software. http://www.blender.org/

[29] Yabee. Yet Another Blender EGG Exporter. https://code.google.com/p/yabee/

Luca Palazzo is a postgraduate research fellow in the Information Engineering Department of Università Politecnica delle Marche (IT). He received a master degree in Informatics Engineering in February 2012 with a thesis entitled "Formalisms to represent software agents mental attitudes and to support their speech act dynamycs".
His research interests include multi-agent systems, machine learning, logic programming and expert systems.