Agents for Serious gaming: Challenges and Opportunities

Frank Dignum

Utrecht University
Contents

• Agents for games?
• Connecting agent technology and game technology
• Challenges
• Infrastructural stance
• Conceptual stance
• Design stance
• CIGA: middleware solution
• Conclusions
Do characters need to be intelligent?
Do we need agents for more serious games?
Agent features (claimed)

1. Goal directed
   - Agents find ways to reach a goal rather than execute a fixed procedure
   - In case of failure of a plan they can replan

2. Reactive behavior
   - Agents react to events in their environment (while keeping their goal in mind)

3. Social abilities
   - Agents know how to communicate in a high and flexible way (ACL is based on speech act theory)
GOAP vs. Agents
GOAP vs. Agents (failing actions)
Goal tree vs. rule based planning

- Goal trees work well to describe default possibilities
- Trees get really messy when incorporating unexpected events and/or failures
- Rules are more suited to cope with these situations
- Divide rules in normal operation rules (default plans) and exception handling rules

- Flexibility comes at the cost of extra specification of general exception handling knowledge (based on domain)
Agents for Games?

- **Assume** that we want to use agents for creating “intelligent” characters in games.

- Can we use Agent Technology to implement those agents in the games?

- I.e. can we make use of all the tools, techniques and platforms that are developed to implement intelligent agents for the incorporation of agents in games?

- If so, what do we need to do to couple the agent and game technologies?

- Or do we have to start from scratch and develop everything again specially for the game environment?
Game Engines and Agents
Client side approach

- Input module
- Game logic/loop
  - Physics Engine
  - Animation Engine
  - Graphics Engine
  - Sound Engine
Example: Pogamut
Multi Agent Systems

Agent Platform

Agent
Agent
Agent

Communication Manager
User Interface

Environment
Environment
Environment
Environment
Game Engines and Agents
Server side approach

Agentified Character

Agentified Character

Agentified Character

Game Engine

Physics Engine

Animation Engine

Graphics Engine

Sound Engine
Example (THOMAS, Aranda et.al.)
Games plus Agents

Input module GE

Agent

Agent Platform

Communication Manager
User Interface

Agent

Agent

Agent

Environment MAS

Game logic/loop

Physics Engine

Animation Engine

Graphics Engine

Sound Engine

Control?
Games plus Agents

Agent Platform

Communication Manager
User Interface

CIGA

Ontological mapping
Dynamic Event subscription
Communication
Event queues

Game logic/loop

Physics Engine
Animation Engine
Graphics Engine
Sound Engine

Agent
Agent
Agent
Agent
Intelligent Virtual Agent Design Issues

- IVA-design is distributed
  - Physical-layer + Cognitive-layer
  - Physical aspects vs. Cognitive aspects
- Cannot design these layers independently
Middleware Approach

- Bridge conceptual gap using a middleware
  - Design problems not responsibility of GE or MAS

- Middleware to provide technical facilities:
  - Translate data representations
  - Perception/action/communication mechanisms

- Don’t restrict designers in their IVA design, but offer technical solutions to help them realizing their design

- Performance determined by how the facilities are used

- Middleware itself is not part of the IVA design!

- CIGA Framework developed to follow this design approach
CIGA Framework

- **Physical Interface**: Connect to simulation environments
  - *E.g.* CORE, (UT, CryEngine, Ogre, Delta3D, etc)
- **Cognitive Interface**: Connect to agent systems
  - *E.g.* Jadex, 2APL, BT-based MAS, etc
- **Connection Mechanism**: Internal message-passing system
  - Introduced for flexibility and portability
  - *E.g.* TCP/IP, Java/C++ bridge
- **Ontology Model**: contract between GE and MAS
  - *E.g.* Specify ontology using: Protégé, custom ontology editors
Connecting the Game engine

- *Physical Interface* integrated into game engine as external component included in the update loop

- Motivation: become less dependent on the (limited) features provided by a particular game engine.

- Offers:
  - Monitoring entity creation
  - Time synchronization
  - Translation world state data to ontological sensory information
  - Perceptual attention: full control (what and when/how often)
  - Behavior realization: framework to implement actions
Connecting the MAS

- *Cognitive Interface*: integrated into MAS as event-based component (no synchronized update)

- Motivation: Provide simple interface for easy integration of wide range of MASs.

- Offers:
  - Notify MAS about possible entities to embody
  - Agent’s sense-act interface where data are instances of ontology concepts
  - Access to ontology model from within the MAS
CIGA Platform + Tools

Features
- Monitor agents
  - Events, actions
  - Subscriptions, logs
- Test actions
- Profile agents
- Inspect ontology model

Run-time Platform GUI

Middleware Configuration

Ontology-editor import scripts

Code Generation Tools

XML
C++
Java
Aspects that make agents work in games

1. Ontology
   • reason on the right abstraction level

2. Perception
   • Get enough and not too much information

3. Action
   • Perform physical actions and react adequately on failure

4. Communication
   • Multi-modal communication
Data representation: Ontology

- Problem: Different data concepts in GE and MAS
  - World state vs. strategic abstraction level
- Solution: Translation-step during agent sensing on GE-side
- Design issue: Suitable abstraction level (not too low, not too high)

<table>
<thead>
<tr>
<th>Conceptual Aspects</th>
<th>Technical Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>- interpretability</td>
<td>- efficiency</td>
</tr>
<tr>
<td></td>
<td>- communication-costs</td>
</tr>
</tbody>
</table>
Ontology Model

- Contract on concepts communicated between GE and MAS

- Designers specify level of abstraction for sensory information and actions based on requirements for specific domain

<table>
<thead>
<tr>
<th>Objects</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Object</td>
<td>location, size</td>
</tr>
<tr>
<td>Human</td>
<td>gender, age</td>
</tr>
<tr>
<td>Fire</td>
<td>type, heat</td>
</tr>
<tr>
<td>FireExtinguisher</td>
<td>type</td>
</tr>
<tr>
<td>Bucket</td>
<td>content, amount</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttackFire</td>
<td>fire, equipment</td>
</tr>
<tr>
<td>Pickup</td>
<td>target</td>
</tr>
<tr>
<td>Communicate</td>
<td>target, message</td>
</tr>
</tbody>
</table>
Ontology: Object Perception Model

- The Object Perception Model defines the ontology into which both the AT and the GE have to map.

Example:
<class name="Character">
  <property>
    <name>ID</name>
    <type>number</type>
  </property>
  <property>
    <name>Distance</name>
    <type>meters</type>
  </property>
  <property>
    <name>Direction</name>
    <type>Orientation</type>
  </property>
  <property>
    <name>Tool</name>
    <type>Tool</type>
  </property>
</class>
Ontology: Interaction model

<Agent name="Door-opener">
  <general>
    <property>
      <name>HoldsOpeningTool</name> <type>Tools</type>
    </property>
  </general>

  <physical>
    <property>
      <name>height</name> <type>meters</type>
    </property>
  </physical>

  <sensor name="eyes">
    <property>
      <name>Range</name> <type>meters</type>
    </property>
  </sensor>

  <capability name="Open door">
    <property>
      <name>target</name> <type>Door</type>
    </property>
  </capability>

</Agent>
Ontology: Interaction model

• PRECONDITION "OpenDoor":
  Poss(OpenDoor(Agent,Door)) ⇔
  Closed(Door) ∧ Distance(Agent,Door)<1 ∧
  Holds(Agent,Axe)

• POSTCONDITION "OpenDoor":
  Done(OpenDoor(Agent,Door)) ⇒
  Open(Door) ∧ Poss(Backdraft(Door))
Control over Perception

- Problem: Perceptual attention for agents
  - Cannot attend to all information from the environment
  - Filtering cannot be performed by GE or MAS alone
- Solution: Subscription-based filtering mechanism
  - Agent controls sensing: what and when to sense
- Design issue: Balance flow of sensory information (not too much, not too little)

<table>
<thead>
<tr>
<th>Conceptual Aspects</th>
<th>Technical Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>- goal-directed/ stimulus-driven</td>
<td>- performance MAS</td>
</tr>
<tr>
<td></td>
<td>- performance GE</td>
</tr>
<tr>
<td></td>
<td>- communication-costs</td>
</tr>
</tbody>
</table>
Perception framework
Implementation
Subscription rules

Example:

\[
\text{Poss(Perceive(Character,ID)) } \iff \\
(Dist(Character,ID) < 150 \land \text{LineofSight(Character,ID)} \land \text{Direction(Character,ID,towards)})
\]
Perception scenario
Control over Action Realization

- Problem: Different nature of actions in typical GE and MAS environments
  - Modality + Duration
- Solution: Action mechanism for body control + feedback channel
  - Dispatch, abort, feedback about status
  - Define actions at functional level
- Design issue: Suitable abstraction-level (not too low, not too high)

<table>
<thead>
<tr>
<th>Conceptual Aspects</th>
<th>Technical Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>- control</td>
<td>- efficiency</td>
</tr>
<tr>
<td>- individuality</td>
<td>- communication-costs</td>
</tr>
</tbody>
</table>
Communication

- Problem: Different communication in MAS and GE
  - Method: communicative intent (direct) vs. verbal and nonverbal communicative behavior (indirect)
  - Communication channel: reliable vs. unreliable
- Solution: Communication mechanism.
  - Allow MAS-communication through simulation environment
- Design issue: Choose method: behavior or intent

<table>
<thead>
<tr>
<th>Conceptual Aspects</th>
<th>Technical Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>- interpretability</td>
<td>- complexity</td>
</tr>
<tr>
<td></td>
<td>- efficiency</td>
</tr>
</tbody>
</table>
Communication is multi-modal
Multi-modal communication
Example rules in modules:

- **PRECONDITION:**
  \[ \text{Poss}(\text{Send}(\text{Propose}(\text{Action},\text{Agent}))) \iff \text{Dist}(\text{Agent}) < 5 \]

- **POSTCONDITION:**
  \[ \text{Done}(\text{Send}(\text{Propose}(\text{Action},\text{Agent}))) \land \text{Dist}(\text{Agent'}) < 5 \implies \text{Poss}(\text{Receive}(\text{Propose}(\text{Action},\text{Agent}))) \]

Can be used to describe physical constraints on communication and side effects of communication.
Communicating agents

<table>
<thead>
<tr>
<th>#</th>
<th>Agent</th>
<th>Activity</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>schedule intent</td>
<td>communicate(d=1, content=inform_child_in_house)</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>schedule action</td>
<td>speech(id=a1, resource=child_in_house.mp3)</td>
</tr>
<tr>
<td>3</td>
<td>S</td>
<td>receive action feedback</td>
<td>action_feedback(action=a1, state=started)</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>perceive action</td>
<td>action_percep(action=a1, state=started)</td>
</tr>
<tr>
<td>5</td>
<td>S</td>
<td>receive intent feedback</td>
<td>intent_feedback(id=i1, state=started)</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>perceive intent (no content)</td>
<td>intent_percep(state=started)</td>
</tr>
<tr>
<td>7</td>
<td>S</td>
<td>receive action feedback</td>
<td>action_feedback(action=a1, state=finished)</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>perceive action</td>
<td>action_percep(action=a1, state=finished)</td>
</tr>
<tr>
<td>9</td>
<td>S</td>
<td>receive intent feedback</td>
<td>intent_percep(id=i1, state=finished)</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>perceive intent</td>
<td>intent_percep(id=i1, state=ended)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Middleware</th>
<th>Activity</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Comm. Facilitator</td>
<td>send intent hint</td>
<td>intent_hint(intent=i1, actions=a1,a2)(state=started)</td>
</tr>
<tr>
<td>B1</td>
<td>Perception Facilitator</td>
<td>send action hint</td>
<td>action_percep(action=a1, state=started)</td>
</tr>
<tr>
<td>B2</td>
<td>Perception Facilitator</td>
<td>send action hint</td>
<td>action_percep(action=a1, state=finished)</td>
</tr>
<tr>
<td>A2</td>
<td>Comm. Facilitator</td>
<td>send intent hint</td>
<td>intent_hint(intent=i1)(state=finished)</td>
</tr>
<tr>
<td>C</td>
<td>Comm. Facilitator</td>
<td>send communication result</td>
<td>communication_result(d=1, observed_by=A)</td>
</tr>
</tbody>
</table>
Designing games with agents: issues

• How intelligent can an agent behave (boundaries):
  • Story line
  • Game rules (including communication)
  • Environment (UI and look and feel)
  • Roles
Design games using OperA

- OperA specifies the boundaries of the behavior of the roles in the game
- OperA indicates landmarks that should be reached that can be used to specify the learning goals
- Agents can fill in the roles in different ways:
  - Scripted character
  - BDI agent
  - ...

OperA example: storyline

- Emergency call
- Create team
- Get to location
- Evaluate situation
- Save victim
- Extinguish fire
- Ambulance
- Expert

start → Emergency call → Create team → Get to location → Evaluate situation → end
**OperA example: Scene**

### Interaction Scene: save victim

<table>
<thead>
<tr>
<th>Roles</th>
<th>Leading_firefighter(1), door_opener(1), fire_extinguisher(1), ambulance(2), victim(3),</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>$\exists H \in \text{people}, \exists T \in \text{victim} \quad \text{perceive}(H,T)$</td>
</tr>
<tr>
<td>Results</td>
<td>$r1 = \forall T \in \text{victim}, \quad \text{safe}(T)$</td>
</tr>
<tr>
<td>Interaction Patterns</td>
<td>$\text{PATTERN}(r1) =$</td>
</tr>
<tr>
<td></td>
<td>${ \text{DONE}(T, \text{at}(H,T)) \text{ BEFORE DONE}(B, \text{secure_area}), \text{ }$</td>
</tr>
<tr>
<td></td>
<td>$\text{DONE}(B, \text{secure_area}) \text{ BEFORE Deadline}_H), \text{ }$</td>
</tr>
<tr>
<td></td>
<td>$\text{DONE}(M, \text{stabilise}(H) \text{ BEFORE Dead}(H)) \text{ }$</td>
</tr>
<tr>
<td></td>
<td>$\text{DONE}(T, \text{transport}_\text{to_ambulance}(H)) }$</td>
</tr>
<tr>
<td>Norms</td>
<td>$\text{PERMITTED}(E, \text{blow_obstacles})$</td>
</tr>
<tr>
<td></td>
<td>$\text{OBLIGED}(M, \text{stabilise}(T) \text{ BEFORE Dead}(T))$</td>
</tr>
<tr>
<td></td>
<td>$\text{OBLIGED} (B, \text{extinguish_fire} \text{ BEFORE transport}(H))$ }</td>
</tr>
</tbody>
</table>
## OperA example: Roles in a game

**Role: leading firefighter**

<table>
<thead>
<tr>
<th><strong>Objectives</strong></th>
<th>Fire_under_control, victims_save</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sub-objectives</strong></td>
<td>{get_to_disaster_location, situation_assessment, plan_of_attack, extinguish_fire, rescue_victims}</td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>Command_team_members, order_ambulance, get_experts</td>
</tr>
</tbody>
</table>
| **Norms** | OBLIGED inform(headquarters,plan_of_attack) BEFORE NOW+10  
            IF DO safe(victim) or DO extinguish(fire) THEN PERMITTED damage(building)  
            OBLIGED ensure_safety(team)  
            OBLIGED safe(victims) BEFORE extinguish(fire) |
Conclusions

- Intelligence by design only
- Several stances needed to cover the connection between games and agents
- Need for a middleware between AT and GE
- CIGA is a principled approach that seems promising
- Infrastructure “easy”
- Conceptual connection is domain dependent
- Design using an OperA like methodology seems promising

- What should be done by the agent and what by the game engine?
- Programming agents?
- What should be intelligent? (pathplanning vs. conversations)
- What agent technology/architecture to use?
  - Existing agent technology is not sufficient or very ad hoc
Agent architectures

**Long Term Memory**
- Doctrine Ruleset
- Goal Hierarchy
- Standards Hierarchy
- Preference Hierarchy
- Agent Memory
- Stress Thresholds
- Decay Parameters

**Generic PMFserv Agent**

- PMF Module Scheduler
- Decision PMFs
- Emotion PMFs
- Perception PMFs
- Stress PMFs

**Blackboard (Working Memory)**
- **Chosen action**
- **Calculated Utilities**
- **Calculated Emotions**
- **Perceived Object List**
- **Need Reservoir Values**
- **Coping style**
- **Stress Reservoir**
- **Physiology Reservoir**

**BDI**
QUESTIONS?