ACTIVE OBJECTS TO DEVELOP COMPUTER GAMES FOR BLIND CHILDREN

Cyrille Bertelle
cyrille.bertelle@univ-lehavre.fr

Antoine Dutot
antoine.dutot@univ-lehavre.fr

Damien Olivier
damien.olivier@univ-lehavre.fr

Guillaume Prévost
guillaume.prevost@univ-lehavre.fr

Université du Havre
25 rue Philippe Lebon, BP 540
76058 Le Havre Cedex, France

KEYWORDS

Active-objects, Multimodality, Visual Disability, Specific Peripherals.

ABSTRACT

The TiM project focuses on creating games for visually impaired or blind children. In this context, the TiM platform is designed to help the creation of such games. For the modeling of games, we used *active objects*. This article deals with the benefits and specificities of this approach. Most game creators will not be computer programmers, and to facilitate the use of our tools (provided under the form of Java programming libraries) we define both a higher level and very simple language, and then above it a graphical tool. This framework provides facilities to create both very simple games when the author has no programming skills, or to develop advanced games for more advanced computer users.

THE TIM PROJECT

The overall aim of the TiM project is to offer visually impaired children the possibility to play with multimedia computer games. They will be intended to severely visually impaired children (blind and partially sighted), with different levels of physical and psychological disability, so that they can use them in an autonomous way, without assistance of a sighted person.

This work will be completed by parallel tasks like an evaluation by educators of children behavior confronted to these games. An evaluation and study of cognitive process and educational potential, in the continuation of this work will be done.

Those studies will be oriented toward visually impaired children's capacity to space and cognitive orientation in the game. They will generate a feedback to the software developers and game content designers in order to improve the games.

THE TIM PLATFORM

The approach is to build authoring tools that allow to conceive games from the ground up, or to adapt existing games, that can be played using specialized or normal devices. The parts developed here are:

game engine The game engine is in charge of running the game, managing active objects, and driving the I/O layer. At this level, only the semantic of the game is described. This means for example that we know that characters exist but we do not know how they will look like to the player. Identically, for inputs we know the player can go left, right, up or down, but we do not know how these orders will be given to it.

I/O layer Its role is double: It transparently outputs a game according to the current hardware of the user, and it must input player orders sent via specialized peripherals to present them to the game engine under a generic form.

game programming language TL (TiM Language) This part allows to program a game more easily than simply using the provided API in Java, providing dedicated constructs.

graphical authoring tool This tool add an higher level tool to develop games. It is limited to predefined games but allows to derive them very quickly and easily.

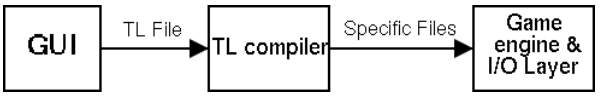

Figure 1: The game creation process

The game engine

Active objects are at the core of the TiM platform. An active object[3] adds a life-cycle to the usual object-oriented approach. It allows them to work in parallel. They own a behavior[2], acting according to rules and beliefs, and "live" in an environment that restrain their acts.

The use of active objects was motivated by a main reason: most of actual games define characters inside

an environment, and active objects directly map on such a concept. Nowadays games uses engines based on this model or close to it. Even games usually implemented sequentially can be easily transcribed using active objects.

In our model, active objects are constrained by an environment that influence their behavior and impose them rules. This is a general view: environments are not necessarily physical. An environment can be 1D, 2D or 3D, in the case of a one dimensional game, the environment is sequential as for example in card games, where it only describe a set of game rules (role, turn, etc.). In 2D or 3D games, at the contrary, the environment is used as a playground in addition to describing game rules.

Active object are not *objects*. This means that active objects are never forced to follow orders of another object. They communicate using messages. They can choose not to respond to a message. They can also behave differently to the same message according to their current environment.

Messages are organized in *streams*. Streams are distinct and every interaction in the system is based on their use. For example there exist streams for vision, or streams for mere inter-object communication, etc. An object never sends the internal representation of what it perceives. It only sends a symbol and the other analyzes it and reacts to it. Some communication streams cannot be ignored by active objects. For example vision streams cannot be canceled. However each object is constrained by its ability to analyze such an input stream. For example some active objects can only see at a given distance.

All in the system is modeled as active objects, comprising the environment. Games or environments are specific derivations of active objects but the base and the relationship between them are the same. This unifies the model.

The I/O layer

To provide games for blind children an input output software has been developed. This part of the platform is then interfaced with a game engine that handles active objects. The input output layer had to be able to use various peripherals. But such devices are often non standard (e.g. braille terminals or sensitive tablets) and the I/O layer must provide *multimodality*[1] to hide this complexity. Multimodality means that it will automatically recognize peripherals and provide the appropriate I/O drivers. For example, both the keyboard arrow keys and a joystick can be used to control a character in a game, or a character can be rendered as sounds for blinds or on a display with high contrast for visually impaired people, and this transparently for the game designer.

The language

Basically the engine is provided as an API (Application Programming Interface) in the Java language. We then designed a very simple language to allow rapid creation of games for people that do not know Java or are not acquainted with computer programming.

The developed language[4] provides specific constructs and directly maps on the active object model defined by the engine. Here are the main entities: a *game* linking several *scenes*, in turn managing several *actors* or *classes*.

The distinction between actors and classes resides in the fact that classes have no actions or perceptions.

An active object of the game engine is represented by an actor of the language. Such an entity has several states, and a behavior. The behavior defines the messages it understands and what to do according to the current situation when these messages arrive. Furthermore, the actor defines two blocks *perception* and *action* that allow it to estimate its situation and act accordingly when not receiving messages.

Scenes define the environment of actors. They are also active objects, but have no perception or action. A scene can be one dimensional, two dimensional, or three dimensional. When defined as a single dimension, a scene describes the steps of the games: turns for players, etc.

A game is defined by a set of scenes and a behavior block that lists the messages that will switch from one scene to another. Like scenes, the game has no action or perception.

Here is a simple example of a player in a labyrinth. The user must find a treasure. We define five entities: a game, a scene, an actor and two classes.

The game only defines the scene. This scene is activated when the game receives the automatically generated `"init"` message. We will send the `"end"` message ourself when the actor will have found the treasure.

```
game
    TreasureHunter
states
    scene laby: Labyrinth;
behaviours
    on "end" do exit; end
    on "init" do activate( laby ); end
end
```

The scene is initialized when it receives the automatically generated `"init"` message. The `(2)` specification creates a 2D scene:

```
scene
    Labyrinth(2)
states
    actor p: Player;
behaviours
on "init" do
```

```
        read_repr( "labyrinth.txt" );
    end
end
```

An actor is created only according to a scene hence the `(Labyrinth)` added after the actor name. It sends the `"end"` message to the predefined `game` entity.

```
actor
    Player(Labyrinth)
behaviours
    perception do
      if same_location( "Treasure" ) do
          game.comm( "end" );
      end
    end
    action do
        player_movement();
    end
end
```

The classes are empty:

```
class
    Wall
end

class
    Treasure
end
```

We provide a rich set of predefined functions like `read_repr()` that creates a 2D environment from a simple description file or a `player_movement()` that change the location of an actor using the I/O layer.

The graphical interface

Above the language, a GUI (Graphical User Interface) has been defined that allows to create predefined kinds of games. It takes care of the game hierarchy (game, environment, active objects), and automatically handles messages (streams), rules and behaviors. It also provide development methods guiding the game author through the creation steps. Figure 2 shows screenshots of it.

CONCLUSION

We developed several arcade games like PacMan, Doom, but also board games like card games that are readily playable. However the platform still needs development in several areas:

- Automatic detection of deadlocks between active objects,
- I/O improvements,
- streams are currently not completely implemented,
- new kind of games should be developed to test the active objects concept,
- improve our game development methodology.

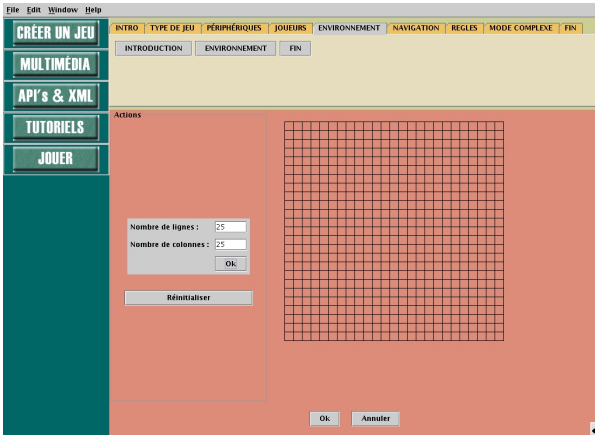

Figure 2: Snapshot of the GUI

ACKNOWLEDGEMENTS

The TiM project is funded by the European Commission, on the program IST 2000 (FP5/ IST/ Systems and Services for the Citizen/Persons with special needs) under the reference IST-2000-25298.

We also thank our TiM partners for their various remarks. More information about the TiM project can be found at:
`http://www.snv.jussieu.fr/inova/tim`.

REFERENCES

[1] D. Archambault and D. Burger, "*TIM (Tactile Interactive Multimedia): Development and adaptation of computer games for young blind children*," in *Proc. ERCIM WG UI4ALL & i3 Sping Days 2000 Joint workshop, Interactive Learning Environments for Children*, (Athens, Greece), Mar. 2000.

[2] Maja J Mataric, *"Behavior-Based Systems: Main Properties and Implications"* in Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, Nice, France, May 1992, 46-54.

[3] P. Terna, *Building Agent Based Models with Swarm*, Journal of Artificial Societies and Social Simulations, 1998.

[4] D. Archambault, A. Dutot, D. Olivier, *TL a Language to Develop Games For Visually Impaired Children*, In Computer Helping People With Special Needs, p. 193-195, ICCHP 2002, Linz (Austria).