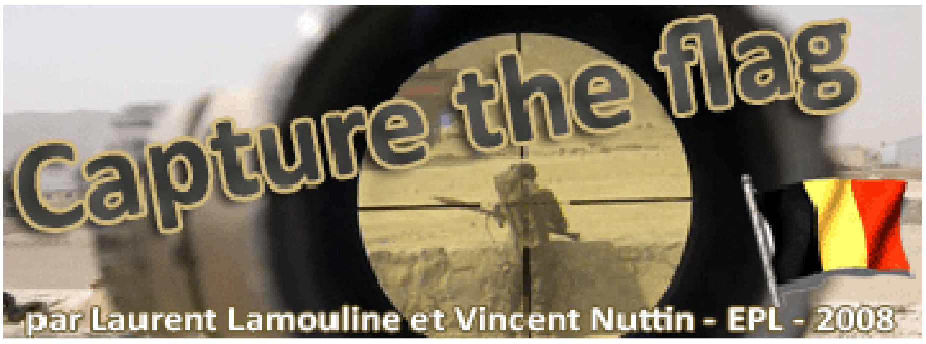
Project : Capture The Flag

INGI 1131 : Concepts of Computer Languages

08/12/2008



Authors: Lamouline Laurent
Nuttin Vincent
Professor: Van Roy Peter
Assistants: Gutierrez Gustavo
Mejias Boriss

Introduction:

As part of lesson INGI 1131: Concepts of Computer languages, we have to implement a game called “Capture The Flag”. This game works with two teams and the aim is, as the name says, to capture the enemies’ flags and bring them back at the team’s home. Players can be caught and killed by other players in the field. They get duplicated when they eat food, and can die by stepping on a bomb. The first team to bring three enemies’ flags back to its base before a certain time wins the match.

Architecture:

There are four major parts in our implementation: the “*player*” which represents a player in the game, the “*environment*”, the “*brain*” and the “*cell*”.
We decided to implement each part named before with *PortObject* because it’s easier to keep the state of each structure and especially to keep track the players’ positions with this. A secondary part is the “*Game*”, it’s a structure that stores as many Cells as there are in the game. It’s the representation of the abstract structure “Environment”. Thanks to this part we can show the game in the GUI. It is the “Environment” that sends messages to the Cell in the Game structure and thanks to the SendToGUI procedure, we refresh the graphical interface.

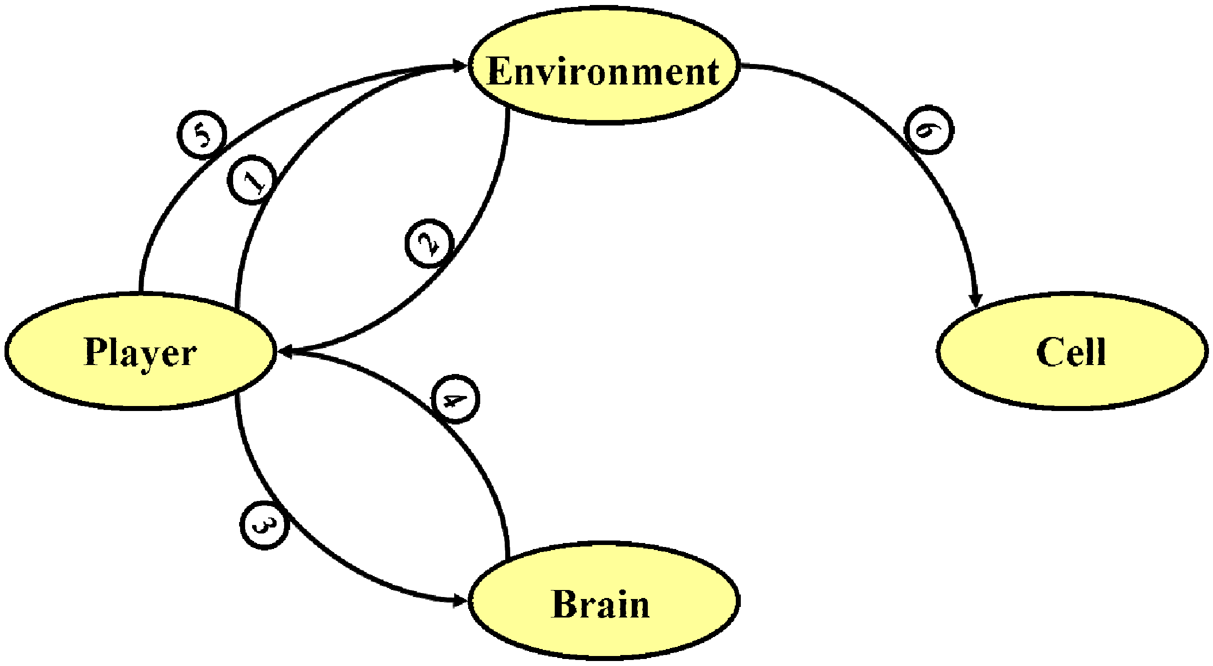


Figure 1 : Execution order

Brief description of a one turns execution:

In the first message (step 1), the player asks the Environment what his neighbors are, that is the nine cells around him. Once he receives the response (step 2), he gives it to the Brain (phase 3). Tanks to a simple AI we implemented, the brain takes a decision in taking account of the cell's content with a priority order we decided (flag -> enemies -> food). Next, the brain sends its decision to the player again (step 4) and then, the player asks the Environment to be moved to the cell the brain chooses (step 5). In the next step (step 6), the Environment sends a message to the Cell in order to move the player to its new position. Finally, the environment actualizes the player's state in order to know its new position, if it carries a flag, …

As said before, we implemented a graphical interface to visualize graphically the players' evolutions. When the state of the cell is changed, a procedure called "SendToGUI" changes the images corresponding to the cells so we can see them on the screen.

Here is a brief presentation of the most important messages "travelling" between the different components (we don't care about the interactions with the "Game", that is we don't take into account the actualization of the graphical pannel):

Player —> Environement:

- ask(nw:N1 n:N2 ne:N3 w:N4 h:N5 e:N6 sw:N7 s:N8 se:N9 posX:X posY:Y)
- movePlayer(name:Name currentX:X currentY:Y team:Team flag:Flag action:Action eFlag:EFlag)

Player —> Brain:

- out(action:Action posX:X posY:Y nw:N1 n:N2 ne:N3 w:N4 h:N5 e:N6 sw:N7 s:N8 se:N9 team:Team flag:Flag homeX:HomeX homeY:HomeY)

Environement —> Player:

- moved(state(name:Name posX:NewX posY:NewY team:T flag:NextFlags eFlag:none))

Environment —> Cell:

- getMagicState(X)

—>: means "… sends to …"

Conclusion:

To conclude, this project was really interesting to learn more about the portobjects and Oz in general, but despite the time spent to make a good game, it still needs adjustments, our program is not perfect, and principally when players interact with each other.
We didn't find a lot of things that do not work, but sometimes (very seldom) players do incomprehensible things and we don't understand why.
We do not pretend to choose the best solution to implement the game but taking into account the time we had to implement such a game, it seems acceptable.