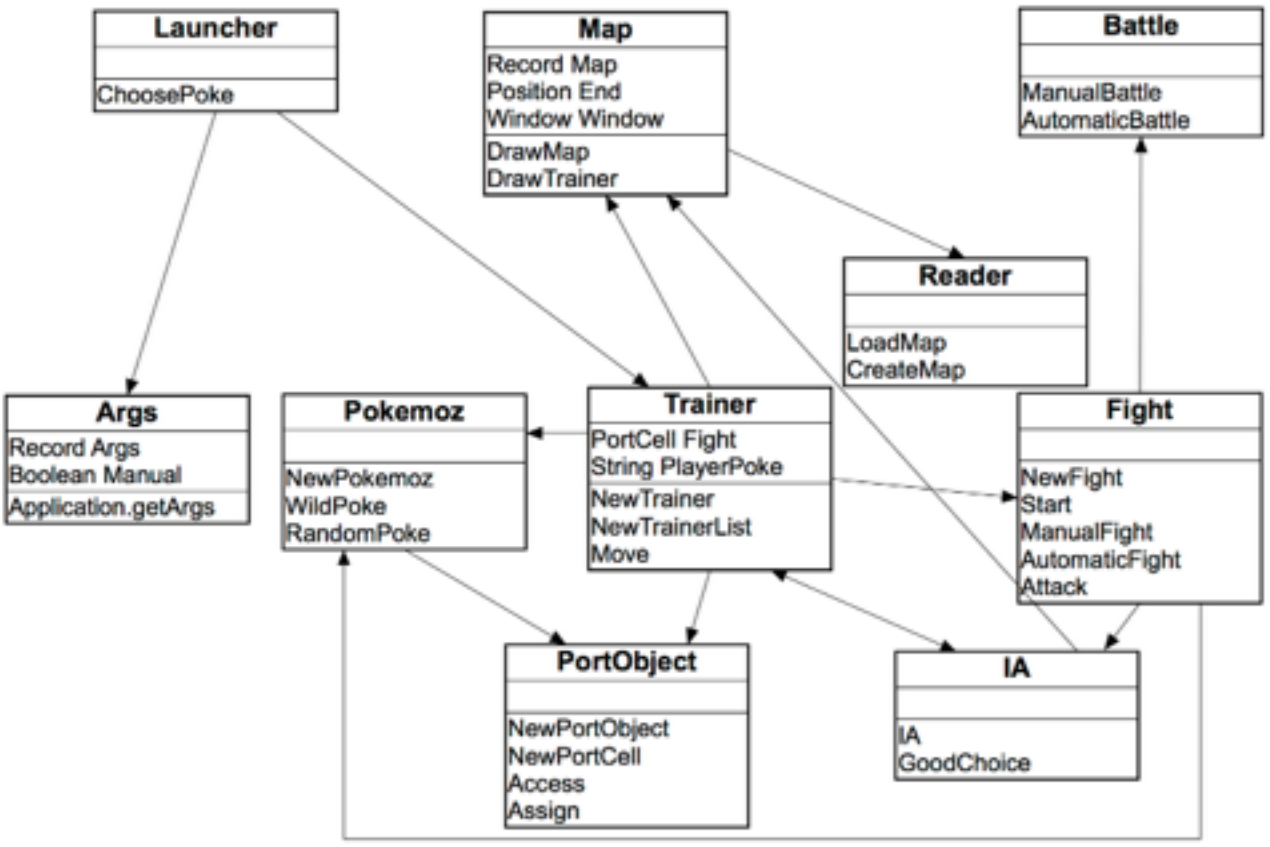
Designing the PokemOZ Project

LINGI 1131
Anne-Sophie Branders - Baptiste Degryse

Architecture of our system

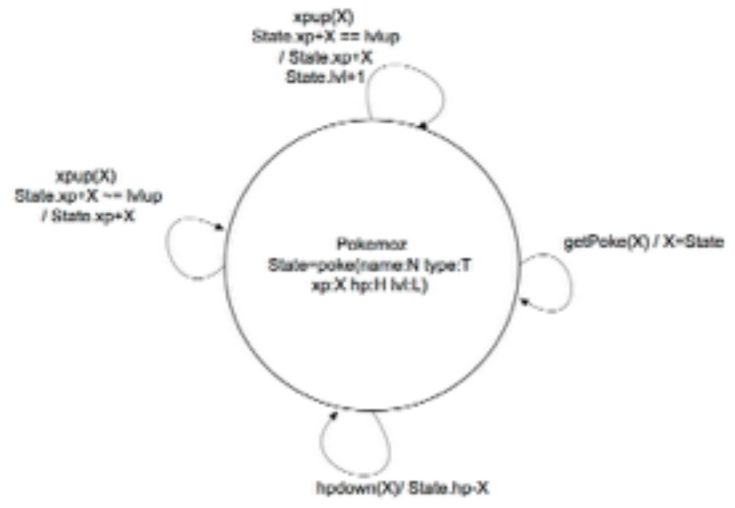
To work on a structured and understandable code, we divided it in different functors. Here is a component diagram to understand the actual structure of our project :



We decided to model the major components by ports objects, as seen in the course. Thus, a component has a state and can receive messages from others. These messages are then processed by a function that depends on the current state of the component and returns his new state. Three major components have been modelled by a port object : **Pokemoz**, **Trainer** and **Fight**. We will now develop them and the other important parts of our project.

Pokemoz

A PokemOZ is represented by its different characteristics : pokemoz(Name Type XP HP Level). During a fight, we will send messages to change its state (if he looses some xp, wins a level, etc). You can see the state diagram of this port object :

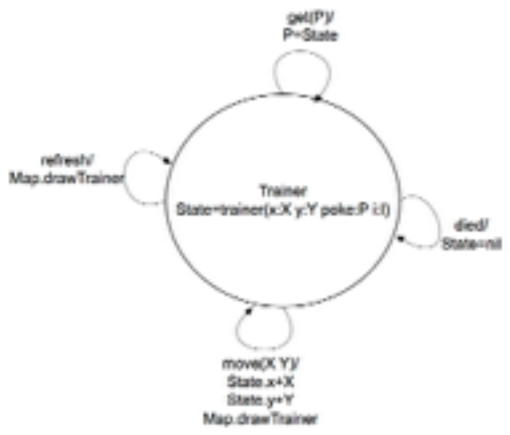


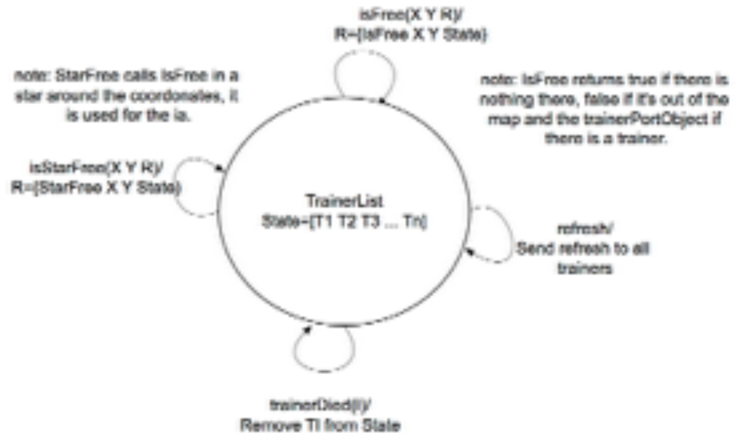
Trainers

A trainer is represented by its actual position on the map, its PokemOZ and its ID : trainer(X Y Pokemoz ID). During the game, its state will always change because trainers are always moving. If our PokemOZ kill a trainer, the state of the trainer will become « died ».

Another port object is also created in the functor « trainer » : a trainer list. It is represented by all the trainers actually living and walking on the map, so it's a list of trainers. We can so easily remove a died trainer from the map or ask them to start moving or to stop.

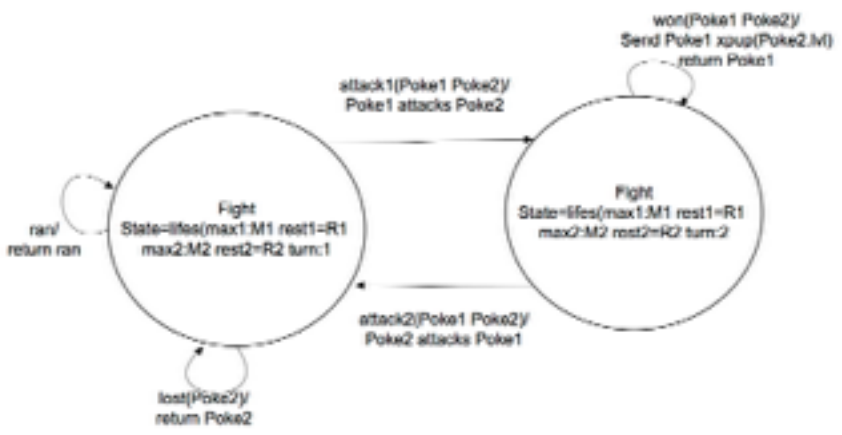
Here are the state diagrams :





Fight

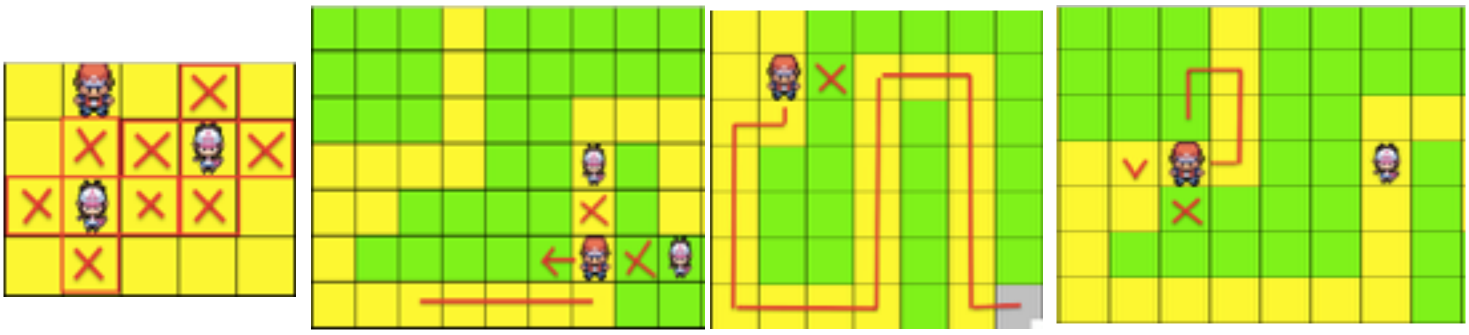
The last port object is the fight. Each time our PokemOZ meets a wild PokemOZ or another trainer, a new fight is created. A fight is represented by the maximum and actual health points of both PokemOZ and the next PokemOZ who can attack : fight(MaxHP1 RestHP1 MaxHP2 RestHP2 Turn). When a PokemOZ dies, the state of the fight becomes the PokemOZ who won the battle. And if we decided to run away from the battle, the state become « ran ». The state diagram is :



All the graphical interface of a battle is in another functor, « battle », to lighten the code.

IA

Another important functor in our program is the artificial intelligence. In fact, if you choose the automatic option, the computer has to take all the decisions in order to try to win the game : the main points of the IA was to reach the end and stay alive. The trainer avoidance isn't perfect, so the trainer must train first. While his PokemOZ isn't level 7 yet, the trainer will stay in the grass so he can choose to run away if the wild pokemOZ isn't the right type or the right level. Then, the trainer will try to reach the end by staying on the road. If he needs to walk in the grass in order to dodge another trainer, he will do so. The trainer cannot come back on one of the last 5 positions in the grass, and 7 positions in the road so he cannot do a repetitive circle of 6 cases. The other trainers will be dodged as if it had a star shape so the next enemy move wont start a fight. If the player goes 5 times to the same position in the last 15 moves, he will change his behaviour and go straight to the end for the next few moves. This IA allows the trainer to adapt himself to the different maps and find the best way to the end.



Conclusion

To conclude this project, we can see that the principal data structures we used are ports. Indeed, it solved the problem of single assignment variables and offered us the possibility to easily transfert messages and variables from one place of the program to another. We also used a lot of records because it offers the possibility to keep together different information of different types.