# Monster Game – Domain Model

The MonsterGame domain model is composed by various small components that, by cooperating with each other, constructs a simple and clean model layer.

The primordial class is Monster, that represents each alive monster in the field. Each monster instance is responsible for asking to move itself around the field and changing its state between protected and vulnerable. This behaviour is achieved through two tasks (Runnable instances) registered in the Monster class. These two tasks are added to a ThreadPool offered by MonsterGame class. They need to be ran in the UI Thread, and Monster class provides a template method with this purpose, runOnSpecificThread, that receives a function as argument and must be implemented when Monster is instantiated.

The MonsterCell class represents each cell in the field (a matrix of n x m cells) in which a monster can stay. When a monster wants to move, it calls the method moveToRandomAdjacentCell() of the current cell it occupies. If there is no available (empty) adjacent cell to move, the monster doesn't move and wait for the next time it can try.

The MonsterField class represents all the positions where a cell could exist, and consequently, a monster could exist. Number of rows and columns need to be passed as arguments when instantiating this class.

Finally, MonsterGame is the class that represents the game. Its responsibilities goes through starting a new game or cancel a running one, compute the time spent to kill the monsters etc. This class has a listener to communicate with the Controller layer (allowing view updates).

By implementing this layer as we did, we tried to remove any reference to other layers, mainly to the View layer with the intention of make this model as reusable as possible, independent of the view used. We hope we have achieved our intent.