Multi-agent systems
Behaviour simulation
Individual-centered approach

Authors :
Salla DIAGNE
Anis TELLO

https://github.com/sallareznov/scalagent

10th FEBRUARY 2016

Table of contents

1 Background **4**
1.1 Presentation . . . 4
1.1.1 Particles . . . 4
1.1.2 Wator . . . 4
1.1.3 Hunt . . . 5
1.2 Technical environment . . . 5
1.3 Utilisation . . . 5

2 Technical work **7**
2.1 Architecture . . . 7
2.1.1 Core . . . 8
2.1.2 Particles . . . 9
2.1.3 Wator . . . 10
2.1.4 Hunt . . . 11
2.2 Behavior . . . 12
2.2.1 Particles . . . 12
2.2.2 Wator . . . 12
2.2.3 Hunt . . . 13

3 Performance **14**
3.1 Particles . . . 14
3.2 Wator . . . 14
3.3 Hunt . . . 14

Références **15**

Chapter 1

Background

1.1 Presentation

This project gathers three multi-agent systems with forms of intelligence in order to simulate different behaviors.

1.1.1 Particles

This application simulates a chamber of particles, meaning a finite environment in which particles moves in their own direction and that can collide with other particles or walls of the chamber. The application never stops.

1.1.2 Wator

This project simulates a game of life in a finite environment with preys (tunas) and predators (sharks). A tuna can only die if eaten by a shark. A shark can only die if it didn't eat after a certain period of time. Tunas and sharks can all reproduce.

The balance of this ecosystem is very delicate: the populations of two species can follow hugely different cycles depending on the given parameters (such as reproduction cycles and the time period in which a shark must eat to avoid starvation) as well as starting positions of each being. We may go from both species being endangered to an abundance of one or both.

When the prey are numerous, predators can reproduce rapidly. But this increase in turn increases the number of prey hunted and the population of the prey decreases. By becoming rarer prey, predators begin to starve and die of starvation, decreasing their population and easing the pressure on hunting prey. The prey (and in time predator) can then go back to rapidly reproducing as the cycle repeats itself.

1.1.3 Hunt

This project is also a predator-prey system. The prey is guided by the user's gestures on the keyboard and predators use Dijkstra's algorithm to find the shortest path leading to the prey [1]. Unlike sharks in the wator environment that only eat tunas nearby, the predators have a strategy. Obstacles are also agents but they do not have any intelligence; they just occupy space. The application stops if the prey has been caught by one of the predators.

1.2 Technical environment

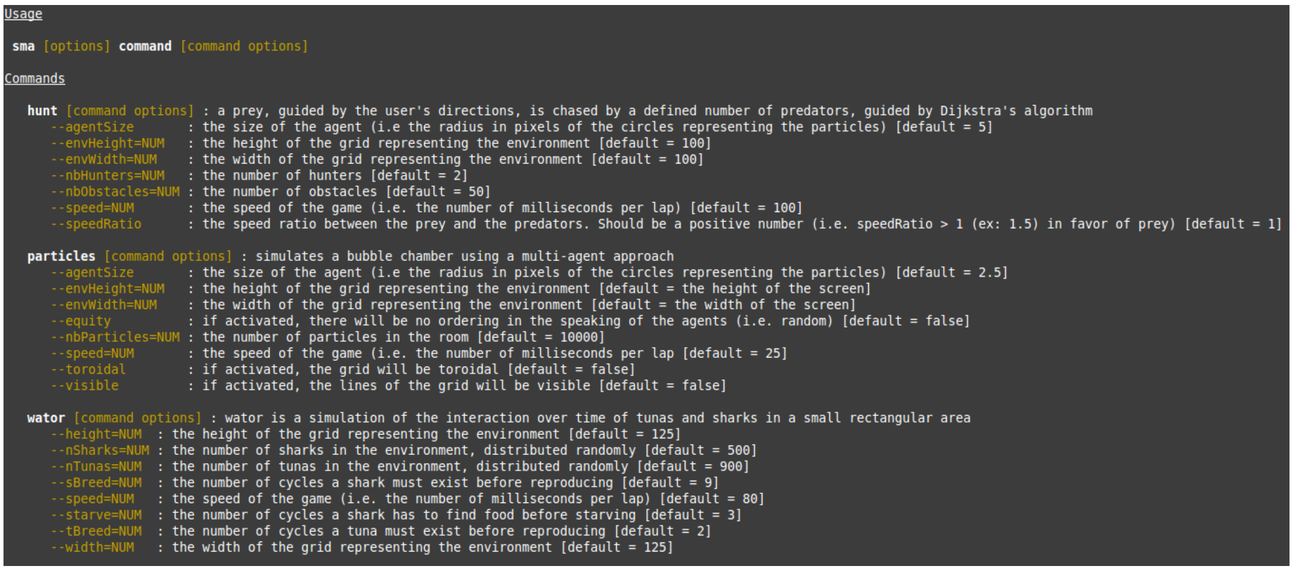
The project has been developed in the Scala language, using the graphical library ScalaFX to render the views. The project uses SBT (Scala Build Tool) as a build system. All the views in the three applications are canvas, and objects are drawn inside.

The source code of the project can be browsed at this GitHub repository :
`https://github.com/sallareznov/scalagent`

1.3 Utilisation

The project is very easy to use. The three applications can be launched from only one executable archive. To compile the project, use the command `sbt assembly`. This will generate an executable archive named `scalagent.jar` in the folder `target/scala-2.11/`.

The usage of the program is the following:



```
Usage

 sma [options] command [command options]

Commands

   hunt [command options] : a prey, guided by the user's directions, is chased by a defined number of predators, guided by Dijkstra's algorithm
      --agentSize       : the size of the agent (i.e the radius in pixels of the circles representing the particles) [default = 5]
      --envHeight=NUM   : the height of the grid representing the environment [default = 100]
      --envWidth=NUM    : the width of the grid representing the environment [default = 100]
      --nbHunters=NUM   : the number of hunters [default = 2]
      --nbObstacles=NUM : the number of obstacles [default = 50]
      --speed=NUM       : the speed of the game (i.e. the number of milliseconds per lap) [default = 100]
      --speedRatio      : the speed ratio between the prey and the predators. Should be a positive number (i.e. speedRatio > 1 (ex: 1.5) in favor of prey) [default = 1]

   particles [command options] : simulates a bubble chamber using a multi-agent approach
      --agentSize       : the size of the agent (i.e the radius in pixels of the circles representing the particles) [default = 2.5]
      --envHeight=NUM   : the height of the grid representing the environment [default = the height of the screen]
      --envWidth=NUM    : the width of the grid representing the environment [default = the width of the screen]
      --equity          : if activated, there will be no ordering in the speaking of the agents (i.e. random) [default = false]
      --nbParticles=NUM : the number of particles in the room [default = 10000]
      --speed=NUM       : the speed of the game (i.e. the number of milliseconds per lap [default = 25]
      --toroidal        : if activated, the grid will be toroidal [default = false]
      --visible         : if activated, the lines of the grid will be visible [default = false]

   wator [command options] : wator is a simulation of the interaction over time of tunas and sharks in a small rectangular area
      --height=NUM  : the height of the grid representing the environment [default = 125]
      --nSharks=NUM : the number of sharks in the environment, distributed randomly [default = 500]
      --nTunas=NUM  : the number of tunas in the environment, distributed randomly [default = 900]
      --sBreed=NUM  : the number of cycles a shark must exist before reproducing [default = 9]
      --speed=NUM   : the speed of the game (i.e. the number of milliseconds per lap) [default = 80]
      --starve=NUM  : the number of cycles a shark has to find food before starving [default = 3]
      --tBreed=NUM  : the number of cycles a tuna must exist before reproducing [default = 2]
      --width=NUM   : the width of the grid representing the environment [default = 125]
```

Figure 1.1: Usage

For example, if the user wants to launch the hunt application with default parameters, the command to enter is the following:

```
$ java -jar scalagent.jar hunt
```

If the user wants to use the particles applications with a toroidal grid of 15000 particles, the command to enter is the following:

```
$ java -jar scalagent.jar particles --nbParticles=15000 --toroidal=true
```

PS: For the wator application, the time-dependent number of tunas and sharks and the age pyramid are drawn in real time, during the execution of the application.

Chapter 2

Technical work

2.1 Architecture

Globally, the architecture of the project follows the MVC (Model-View-Controller) [2] paradigm. Its fine granularity favorite the Single-Reponsibility Principle . The project is divided in four packages:

```
.
├── core
├── hunt
├── particles
└── wator
```

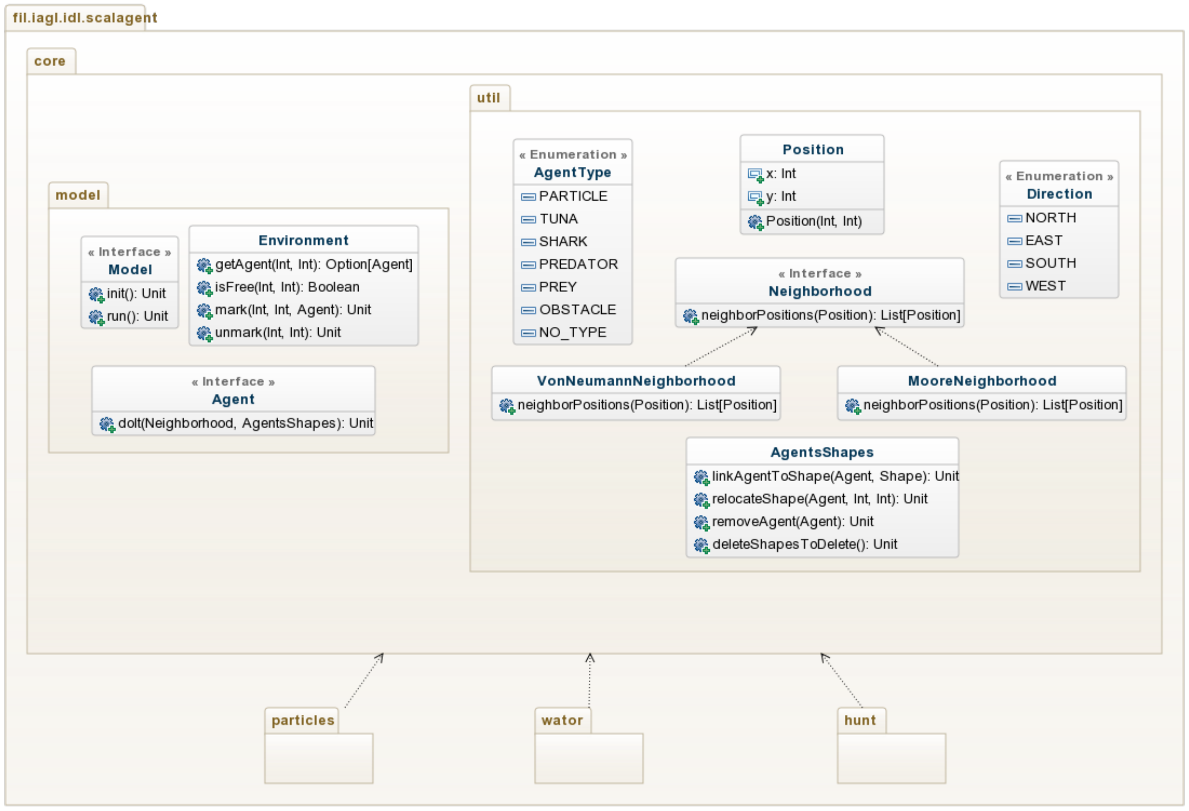
core contains the logic which is common between the three applications

hunt contains the logic related to the hunt application

particles contains the logic related to the particles application

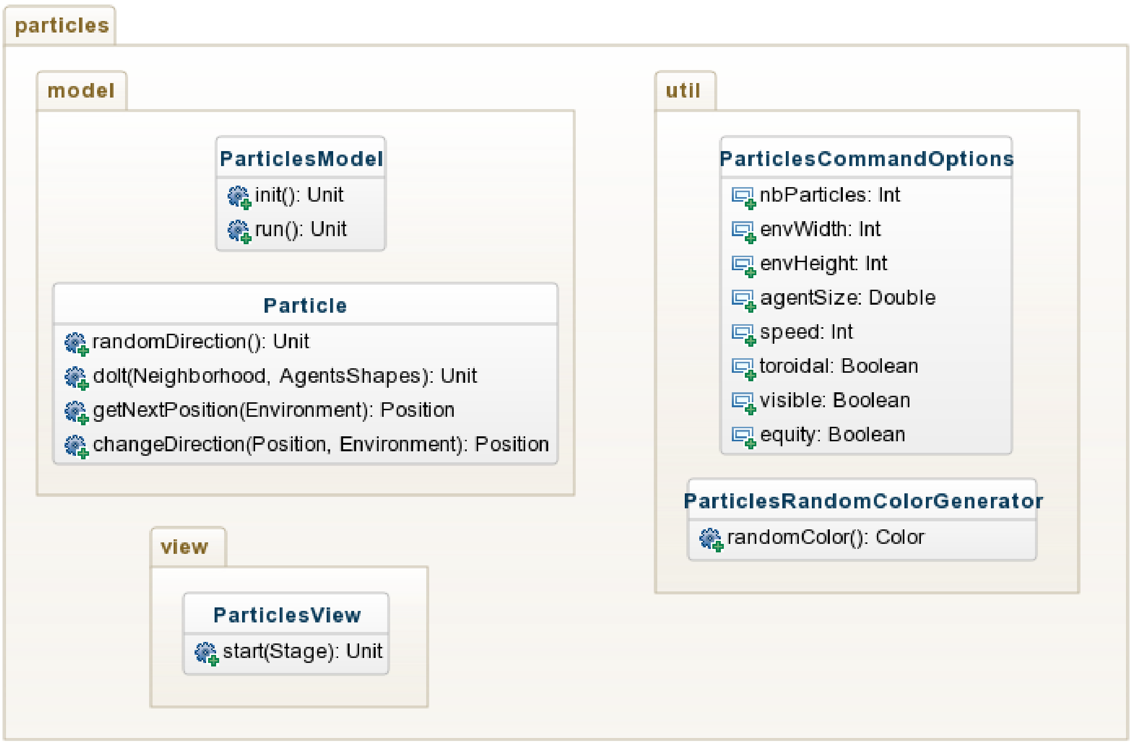
wator contains the logic related to the wator application

2.1.1 Core



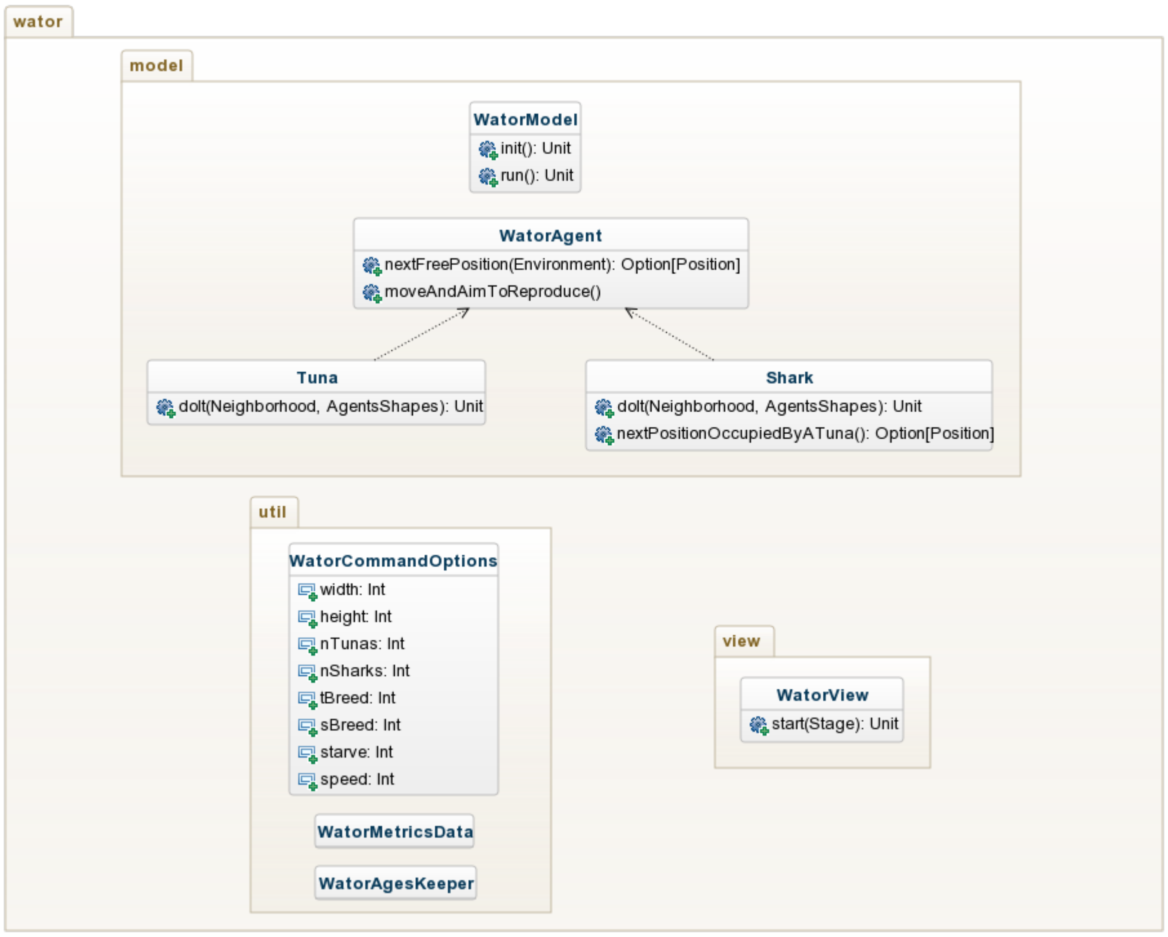
```
.
├── model
│   ├── Agent.scala :  an agent in a multi-agent system
│   ├── Environment.scala :  an environment in the multi-agent system
│   └── Model.scala :  the model of the application within the MVC paradigm
└── util
    ├── AgentsShapes.scala :  the shapes representing the agents
    ├── AgentType.scala :  the type of an agent
    ├── Direction.scala :  the direction of an agent
    ├── MooreNeighborhood.scala :  the Moore neighborhood of an agent
    ├── Neighborhood.scala :  the neighborhood of an agent
    ├── Observable.scala :  the observable within the Observer pattern
    ├── Observer.scala :  an observer neighborhood within the Observer pattern
    ├── Position.scala :  the position of an agent
    └── VonNeumannNeighborhood.scala :  the Von Neumann neighborhood of an agent
```

2.1.2 Particles



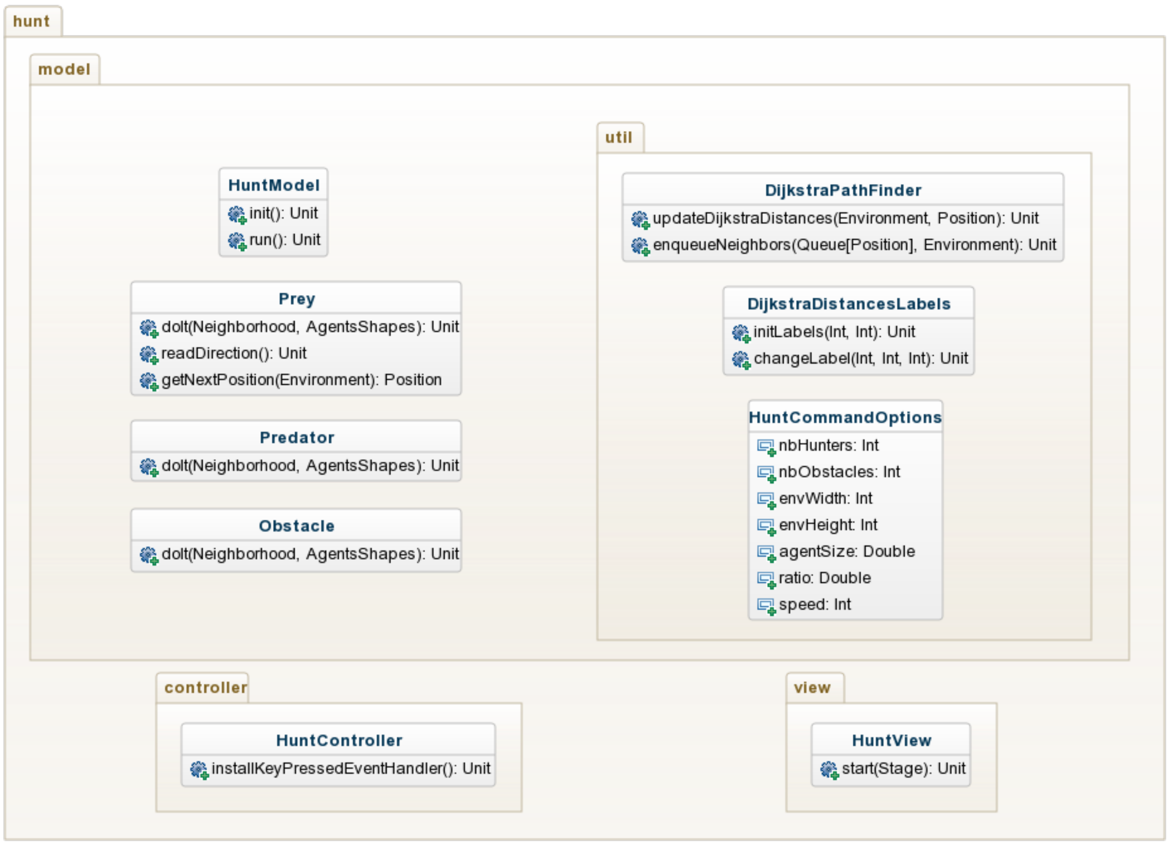
```
.
├── model
│   ├── Particle.scala :  a particle
│   └── ParticlesModel.scala :  the model of the particles application within the
│       MVC paradigm
├── util
│   ├── ParticlesCommandOptions.scala :  the options that can be applied to the
│   │   particles application
│   └── ParticlesRandomColorGenerator.scala :  a random color generator for
│       particles
└── view
    └── ParticlesView.scala :  the view of the wator application within the MVC
        paradigm
```

2.1.3 Wator



```
 model
  ├─ Shark.scala :  a shark
  ├─ Tuna.scala :  a tuna
  ├─ WatorAgent.scala :  a agent in the wator environment (an agent that can
  │  reproduce)
  └─ WatorModel.scala :  the model of the wator application within the MVC
     paradigm
├─ util
  ├─ WatorAgesKeeper.scala :  data for the age pyramid
  ├─ WatorCommandOptions.scala :  the options that can be applied to the wator
  │  application
  └─ WatorMetricsData.scala :  data for the time-dependent number of tunas and
     sharks
└─ view
  └─ WatorView.scala :  the view of the wator application within the MVC
     paradigm
```

2.1.4 Hunt


hunt
model
util
HuntModel
init(): Unit
run(): Unit
Prey
doIt(Neighborhood, AgentsShapes): Unit
readDirection(): Unit
getNextPosition(Environment): Position
Predator
doIt(Neighborhood, AgentsShapes): Unit
Obstacle
doIt(Neighborhood, AgentsShapes): Unit
DijkstraPathFinder
updateDijkstraDistances(Environment, Position): Unit
enqueueNeighbors(Queue[Position], Environment): Unit
DijkstraDistancesLabels
initLabels(Int, Int): Unit
changeLabel(Int, Int, Int): Unit
HuntCommandOptions
nbHunters: Int
nbObstacles: Int
envWidth: Int
envHeight: Int
agentSize: Double
ratio: Double
speed: Int
controller
HuntController
installKeyPressedEventHandler(): Unit
view
HuntView
start(Stage): Unit

```
.
├── controller
│   └── HuntController.scala :  the controller of the hunt application within the
│       MVC paradigm
├── model
│   ├── HuntModel.scala :  the model of the hunt application within the MVC
│   │   paradigm
│   ├── Obstacle.scala :  an obstacle
│   ├── Predator.scala :  a predator
│   └── Prey.scala :  a prey
├── util
│   ├── DijkstraDistancesLabels.scala :  the Dijkstra distances labels drawing
│   ├── DijkstraPathFinder.scala :  the Dijkstra path finder
│   └── HuntCommandOptions.scala :  the options that can be applied to the hunt
│       application
└── view
    └── HuntView.scala :  the view of the hunt application within the MVC
        paradigm
```

2.2 Behavior

2.2.1 Particles

Our particles behave as following: Each particle has directions on X-axis and Y-axis (-1, 0, +1). At each round, each particle will check if the next possible in the given neighborhood (Moore's or Von Neumann's), and according to its direction is free. If it is, it moves to that position. If not (i.e. it clashed with another particle in the environment), it randomly choose an empty position in the given neighborhood.

The method responsible of getting the next position checks whether the environment is toroidal or not.
If the environment is not toroidal, the next possible position if a particle reaches the wall would be the next position after multiplying the direction by -1.
Otherwise if the environment is toroidal, then particles can get throw walls and next possible position would be the first position on the other side of the environment.

At the beginning, All particles are distributed randomly in the environment with random directions.

2.2.2 Wator

WatorAgent (a tuna or a shark) is an agent that has an age and breed counter. Each WatorAgent can move randomly one step at a time in a toroidal environment. Also WatorAgent can breed if its breed counter has reached the fertility time (the number of rounds an agent must exist before reproducing).

Tuna

Tunas agents do not die if they haven't been eaten. At each round, tunas try to move randomly within Moore's neighborhood - if there is an empty position. If it is the case, a tuna will try to reproduce if it can, by moving to the next empty position, and leaves a child at its old position if it can reproduce. If there are no empty place around, tuna do not move and do not breed.

Shark

A shark has a starvation counter (i.e. the number of cycles it can find food before starving). Sharks can die if they reach their starvation point. The first thing a shark will do is to check if it has reached the starvation point and die.
If not, a shark looks around within Moore's neighborhood if there is a tuna. If yes, it will move

and eat it. At the same time, if a shark can reproduce, it will reproduce by moving to the next position occupied by a tuna and leaving a new shark at its old position.
If there are no near tuna, a shark will try to move randomly within Moore's neighborhood and reproduce if it can.
If there are no empty place around then the shark will stay in its place and won't breed.

2.2.3 Hunt

Prey

Prey is an agent that has directions on X-axis and Y-axis (-1, 0, +1) in a toroidal environment. The prey keeps on moving one or more step at a time (it depends on the parameter *speed ratio between prey and predators*) in the same direction. The prey is guided by the user's gestures on the keyboard : he can change the direction of a prey by pressing up, down, left or right button on the keyboard.
If a prey reaches an obstacle it stops (direction on both X and Y axis becomes 0).

Predator

Predators are agents that move within Moore's neighborhood in a torodial environment. Predators use Dijkstra's algorithm [1] to choose a next position in order to catch the prey.

Obstacle

Obstacles are agents that do nothing besides occupying places inside the environment.

Chapter 3

Performance

Globally, the project performs very well and handles the scalability imposed by the different parameters.

The applications has been tested using a 64-bits computer, with 8GB of RAM and running Intel(R) Core(TM) i5-4210H CPU @ 2.90GHz processor. The global performance may vary depending on the machine it has been tested on.

3.1 Particles

We can go up to more than 37000 particles and the application will run smothly.
More than that the application will start quickly and run normally, but refreshing time will start to be visible to the naked eye.

3.2 Wator

For the default environment size (180 x 180) our application can go up to 16000 agents (both sharks and tunas). Default environment size is recommended so both *time-dependent number of fishes* chart and *population chart of fishes* will be rendered correctly.

3.3 Hunt

We didn't reach a performance limit with the hunt application.

Bibliography

[1] Wikipedia. Dijkstra's algorithm. `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`.

[2] Wikipedia. Model-view-controller. `https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`.