Mircea Maria-Madalina, ICA, 246/2

Project 2 – Documentation

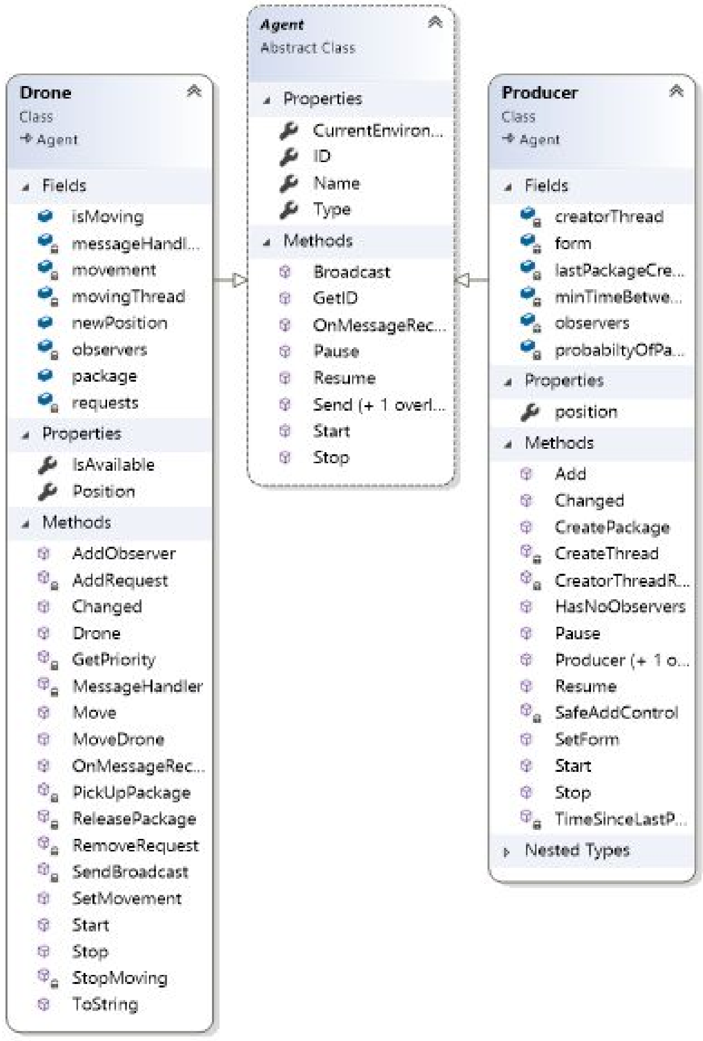
Description of the project: A 2D drone delivery system. Several producers create packages that need to be delivered to a specific address. Drone agents travel to pick up and deliver the package.

Language: C#

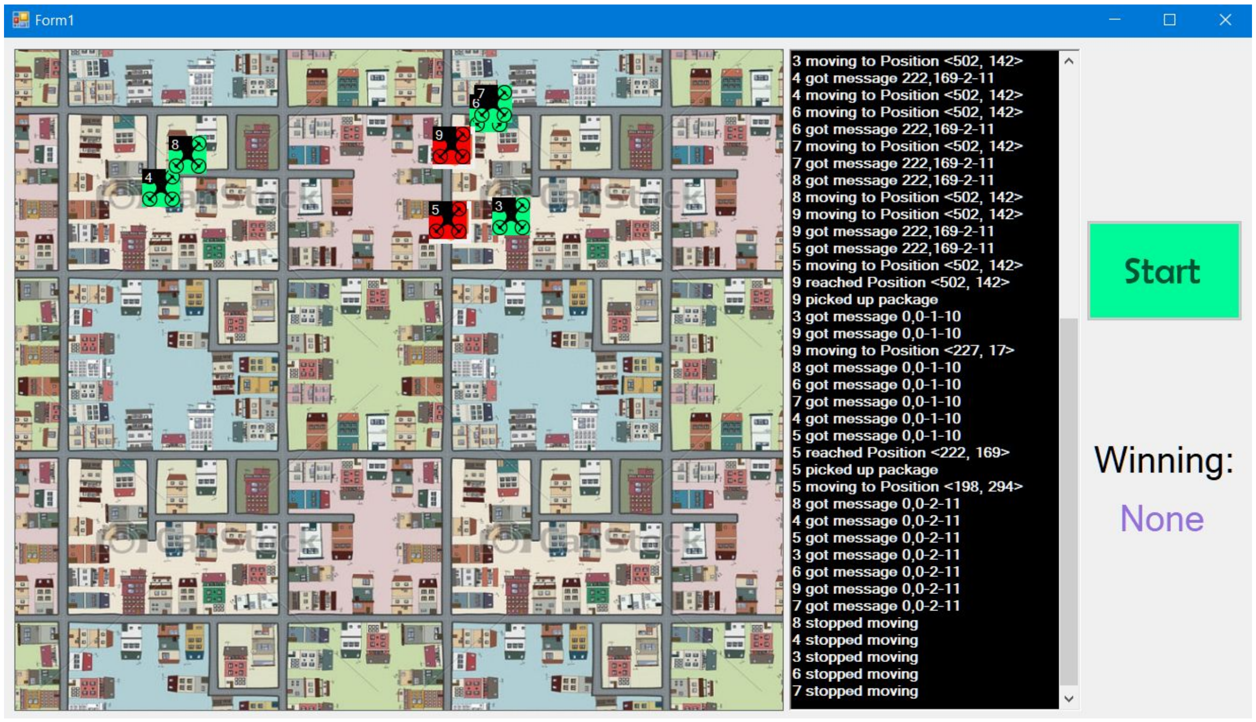
Tools: Visual Studio, Windows Forms

Flow:

- The producers and the drones are implemented using the Observer design pattern. The producers are observables, while the agents are observers.
- The Drone and Producer classes implement the abstract class Agent and override some of its methods (see the figure below for the class diagram)



- The agents communicate between them and with the producers through the environment, with an AgentBlackboard object, using the AgentMessage class.
- The windows forms display shows the map, the agents and packages, but also a console that shows every action that has been performed, for a better understanding of the environment and communication between agents. The display looks something like this:



```
3 moving to Position <502, 142>
4 got message 222,169-2-11
4 moving to Position <502, 142>
6 moving to Position <502, 142>
6 got message 222,169-2-11
7 moving to Position <502, 142>
7 got message 222,169-2-11
8 got message 222,169-2-11
8 moving to Position <502, 142>
9 moving to Position <502, 142>
9 got message 222,169-2-11
5 got message 222,169-2-11
5 moving to Position <502, 142>
9 reached Position <502, 142>
9 picked up package
3 got message 0,0-1-10
9 got message 0,0-1-10
9 moving to Position <227, 17>
8 got message 0,0-1-10
6 got message 0,0-1-10
7 got message 0,0-1-10
4 got message 0,0-1-10
5 got message 0,0-1-10
5 reached Position <222, 169>
5 picked up package
5 moving to Position <198, 294>
8 got message 0,0-2-11
4 got message 0,0-2-11
5 got message 0,0-2-11
3 got message 0,0-2-11
6 got message 0,0-2-11
9 got message 0,0-2-11
7 got message 0,0-2-11
8 stopped moving
4 stopped moving
3 stopped moving
6 stopped moving
7 stopped moving
```

- A producer creates a package. This is displayed on the screen using the following icon:

- The producer notifies the observing agents about the parcel.
- The agents are represented by the following icon:

- The icon has a green background when the agent is available and a red background when it is delivering a package.
- The available agents (i.e. they are not currently delivering a package) race to the new package.

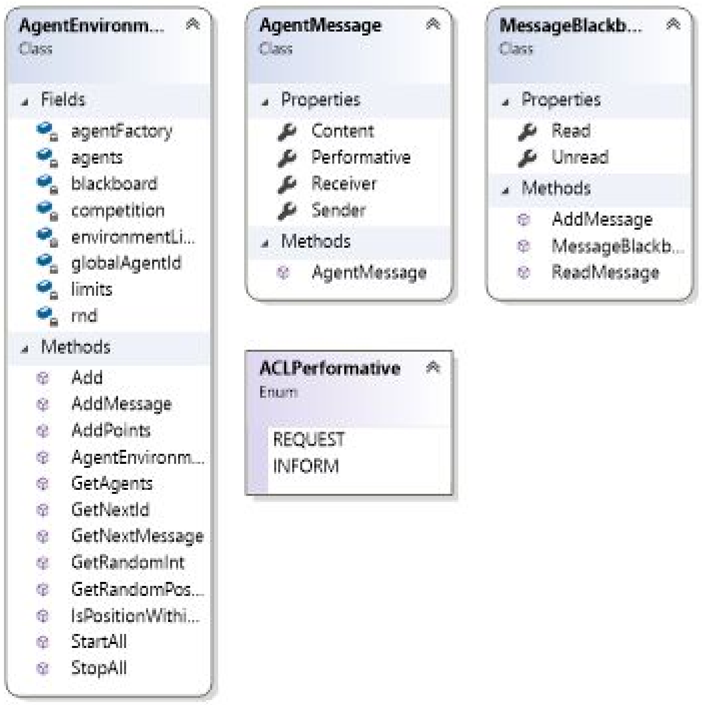
- The agent that is first to arrive gets to deliver the package. The winning agent picks up the package and moves it to the new address. When it arrives at the new address, the package icon disappears and the agent is available to deliver another package.

PAGE:

Agent	Perception	Actions	Goals	Environment	State
Drone	**Delivery requests** **Other drone achievements**	**Move** **Pick up package** **Drop off package** **Compete with other drones**	**Deliver as many packages as possible**	**Image plane (map)** **Map limits**	**Current delivery requests**
Producer		**Create a package**	**Create packages**	**Image plane (map)**	

Messaging:

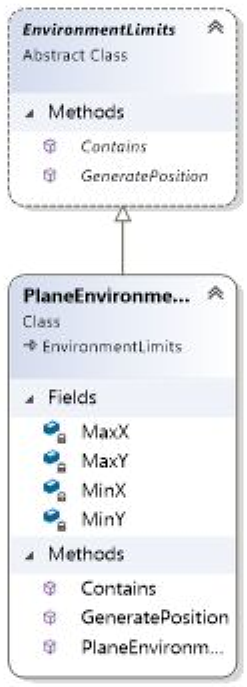
- Blackboard approach - the environment contains an AgentBlackboard attribute which is used to store the shared messages
- The AgentMessage class is used to send messages
- The messaging follows the ACL standard because
  - It has a Performative attribute which is of type ACLPerformative
  - The ACLPerformative enum contains the types
    - INFORM - drones communicate with other drones
    - REQUEST - producers communicate with drones
  - It has the Content, Sender and Receiver attributes



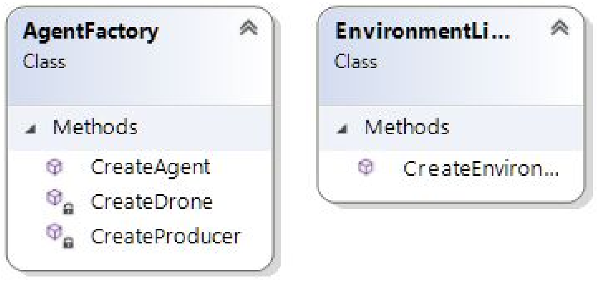
- Each agent has a separate message-handling thread that continuously checks for new messages directed to the agent and reacts accordingly when one such message is received.
- The AgentBlackboard object keeps a “Read” and an “Unread” list. When a message is first added, it is added to the Unread list. When an agent’s message-handling thread reads the message, it is removed from this list and added to the Read list.

Other conceptual and implementation aspects:

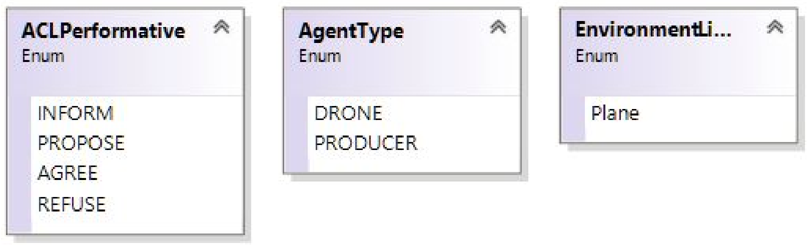
- In order for the tool to be as abstract and general as possible, the AgentEnvironment class contains:
  - an EnvironmentLimits attribute which stores the ”edge” of the environment (minimum and maximum X and Y for a 2D environment) → can be utilized for more than 2 dimensions with minimum changes, compares a given position with the limits and generates random positions within the limits without the need for the environment to be aware of the dimensionality of the space



- an EnvironmentLimitsFactory attribute which is used to create the EnvironmentLimits attribute using the Factory pattern
- an AgentFactory attribute which is used to create agents with the Factory pattern

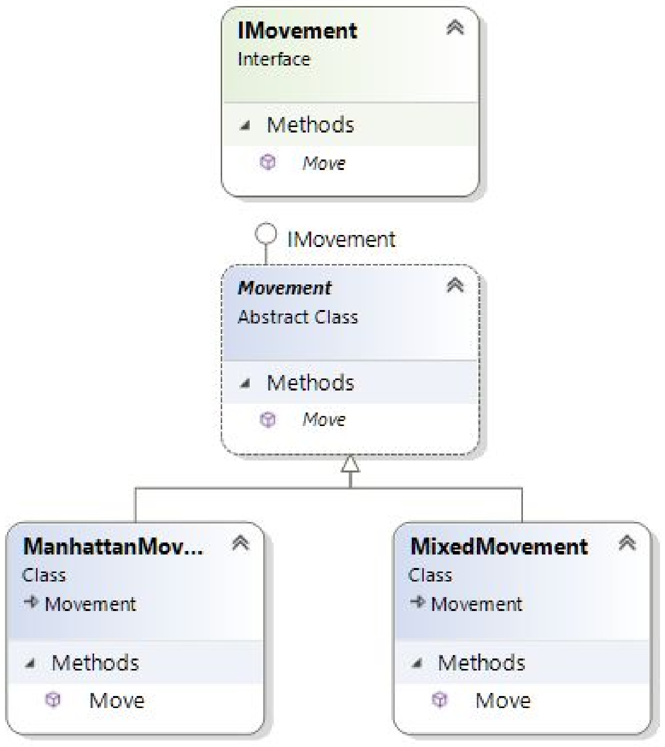


- The factories and messages use the following enumerations:

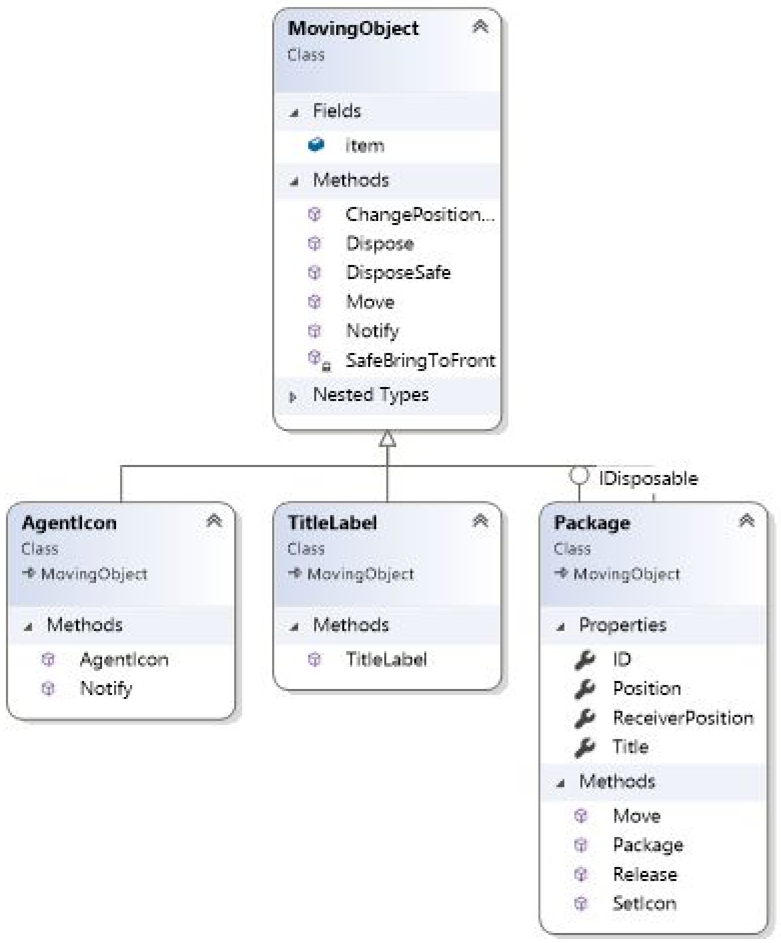


- a Random variable so that the random values used within an Environment come from the same source
- a globalAgentId integer variable used to give a unique ID to every agent

- The drones have two different types of movement: Manhattan movement and Mixed movement. This is decided when the drone is created through a random variable.
    - The drone has a 25% chance to move with the Manhattan movement because it is significantly slower than the Mixed movement.
    - Manhattan movement only moves the drone vertically and horizontally.
    - Mixed movement moves the drone diagonally as well.



    - At a later time, the implementation could be changed to utilize a Strategy design pattern and change this movement periodically, to have variation in a drone's behavior.
- The "front end" is separated from the "back end", in that the drone icons do not depend on the implementation of the drones.
    - The drone icons and labels simply receive an agent to follow and they are notified whenever the agent moves (with the Observer pattern - the drone is an Observable and the icons and labels and Observers). Thus, the application can be started without seeing drones moving on the screen, but with the information about them still visible in the console.
    - The icons and labels (as well as the Package icons) extend the MovingObject class, which is what helps them move smoothly and remain on top of the other visual items in the Form (see the figure below for the class diagram). This is done using safe delegates.


MovingObject
Class
Fields
item
Methods
ChangePosition...
Dispose
DisposeSafe
Move
Notify
SafeBringToFront
Nested Types
IDisposable
AgentIcon
Class
MovingObject
Methods
AgentIcon
Notify
TitleLabel
Class
MovingObject
Methods
TitleLabel
Package
Class
MovingObject
Properties
ID
Position
ReceiverPosition
Title
Methods
Move
Package
Release
SetIcon