In order to develop models of such individual behavior and social interaction to understand the complex of an urban environment, we need a theory that accounts for individual as distinct objects, explicitly representing such behaviors. From this comes the idea of agent based modeling. This chapter introduces an individual –agent– based model and reviews some work that has been constructed with this kind of simulation. The later following is a briefly review on object-oriented programming language we use for implementing our project. The last part of the chapter presents the system architecture of our simulation.

3.1 Introduction to an Individual-based Model

Individual-based models are simulations based on the global consequences of local interactions of members of a population (Reynolds, 1987). These individuals might represent plants and animals in ecosystems, vehicles in traffic, people in crowds, or in some case they can be autonomous characters in animations and games. These models typically consist of an environment or framework in which the interactions occur and some number of individuals defined in terms of their behaviors (procedural rules) and characteristic parameters. In an individual based model, the characteristics of each individual are tracked through time. This stands in contrast both to a character in an animated movie, whose action are scripted in advance, and to an ‘avatar’ in a game or virtual reality, whose actions are directed in real time by a human player or participant. Individual-based models are also known as entity or agent based models, and as individual/entity/agent-based simulations. Individual-based models are a subset of multi-agent system. They are distinguished by the fact that each agent corresponds to an autonomous individual in the simulated domain.

Some individual-based models are also spatially explicit meaning that the individuals are associated with a location in geometrical space. Some spatially explicit individual-based models also exhibit mobility, where individuals can move around their environment. For example, a model of people and buildings in an urban simulation. People are mobile individuals, but buildings in the same situation would not be mobile. Spatially explicit models may use either continuous (real value) or discrete (integer valued, grid-like) space.
There is an overlap between an individual-based models and **cellular automata** (CA). Cellular automata are also spatially explicit, grid/tile based, immobile individual-based models. However, CAs are always homogeneous and dense (all cells are identical) while a **discrete individual based model** might occupy only a few grid cells (see Chapter 3.2.3). In CAs, each cell can have various states, and so represent concepts like empty or occupied. The significant difference is whether the simulation is based on a dense and uniform population of the space, as in CA, or based on specific individuals sparsely distributed within the space.

### 3.2 Example Models

Three examples illustrate what are similar and different characteristics among three kinds of individual-based simulations.

#### 3.2.1 Continuous Individual-based Simulation

One of the best known examples of this type of model is "**boid**", an autonomous birdlike agent. In 1986, artificial life theorist Craig Reynolds created a computer program to simulate the flocking behavior of birds *(Figure 3.1)*.

*Figure 3.1: Simulated boid flock avoiding obstacles (Reynolds, 1986)*
Reynolds placed a large number of *boids* into a three dimensional virtual environment. The agents were programmed to follow three simple rules of, what he later called, **steering behavior**:  
1. **Separation**: to maintain minimum distance from other objects in the environment (other agents, as well as obstacles).  
2. **Alignment**: to match velocities with other agents in the neighborhood.  
3. **Cohesion**: to move toward the perceived center of mass of agents in its neighborhood.

![Figure 3.2: Steering Behavior (Reynolds, 1999)](image)

None of these rules says "form the flocks". The rules were entirely local, referring only to what an individual boid could do and see in its own vicinity. If a flock is to form at all, it has to do so from the bottom-up as an emergent phenomenon. And yet flocks do form, every time.

He has extended his concept of steering behavior to the area of entertainment by applying those behavioral rules for autonomous characters in animations and games to perform a wide range of motion behavior. His recent web site (http://www.red3d.com/cwr/steer/gdc99/index.html) presents a collection of simple, common steering behaviors of animals and human which includes seek, flee, pursuit, evasion, path following, obstacle avoidance, wander, and so on. Many steering behaviors can be quite well adapted to describe the way pedestrians move in small urban space.

![Figure 3.3: Demonstrations of Steering Behavior (Reynolds, http://www.red3d.com/cwr/steer/)](image)
3.2.2 Discrete Individual-based Simulation

This example of discrete models demonstrates the idea that a complex composite object can be generated from a collection of simple single objects. In this case, what is being formed is not individual elements but space itself, rather as a settlement can be generated by aggregating a collection of houses (Hillier and Hanson, 1984).

The elementary objects are square cells. The rule of addition of cells is a full face-wise join (Figure 3.4 a), not the vertex join (Figure 3.4 b). In the aggregation process, objects are added randomly to whatever is already aggregated subject to only one restriction; each cell must retain at least one side free from other cells. By the time a hundred cells have been aggregated, this generative process looks something like Figure 3.4 c.

Whatever the sequence of placing objects, this process will result in a dense and continuous aggregate of cells containing a number of void spaces – rather like courtyards – some of which are big and some small. Once again a global form has arisen from a local rule. Although the global pattern emerges from those individual agents that constructed the object, the form of object is not
the product of any “goal-directed” or “human” organization. However, it is far from obvious that this type of explanation for the generation of such forms in terms of human behaviors is complete, though a real case depicted in Figure 3.5 is evident.

3.2.3 Cellular Automata (CA)

There have been many efforts to providing a software platform for multi-agent cellular automata simulations. Swarm, originally developed at The Santa Fe Institute, and StarLogo (Figure 3.6), developed from a programming language for children ‘Logo’ (Resnick, 1994), are such software platforms for the simulation of complex systems. Each is claimed to be a system that helps scientists and researchers focus on research rather than on tool-building. Quite a number of simulations with cellular automata (compared to other approaches) have been conducted to illustrate complexity in urban systems, from the models of urban morphology to the model of pedestrian crowds and flows.

Figure 3.6: Model of termites piling woodchips developed with the use of StarLogo. (Resnick, 1994)
Cellular automata consist of a regular uniform lattice, usually infinite in extent, with a discrete function and state in each cell or site. CA evolves in discrete time steps, meaning a value of a variable of one site is affected by the states of its neighborhood sites at the previous time step (*Figure 3.7*). The variables at each site are updated simultaneously (synchronously), based on the states in their neighborhood at the preceding time step, and according to a definite set of local rules (Wolfram, 1983).

![Cellular Automata](image)

*Figure 3.7: Regular uniform of Cellular Automata*

The example shown here is the model of multiple lanes and separated flows by Victor Blue (see more at http://ulster.net/~vjblue/p2dir.html). CA provides a tractable approach to model pedestrian movement where people are picking their way through a crowd and in some cases pedestrian lanes are formed. The rules of moving are:

1) pedestrians move in two distinct steps forward; lane changes can be made (left-forward-right);
2) they can exchange places when they find themselves nose to nose (one cell apart)
3) they may avoid lanes in which there is an oncoming pedestrian within 8 cells
4) they have a slight bias to move to the right.

As shown in *Figure 3.8* (next page), dynamic multiple lanes begin to form in a model containing rules 1-3. In the jam situation, simulated pedestrians have found to pick one cell in front and follow it; this has been witnessed in a crowded corridor entering Grand Central Station. But when
rule 4 is added to the model, pedestrians start to drift into opposing same-direction lanes. After a few minutes, separated flow is recognized. Directional lane seeking is an emergent property of the model that is not directly built into it, and it is also an observed phenomenon in real pedestrian behaviors.

![Image of multiple lanes and separated flows](image)

*Figure 3.8: The model of multiple lanes and separated flows (Blue, 1999)*

**Reflection**

Our interest in an individual-based model began when we were looking for a system that can be used for simulating complex pedestrian spatial behaviors in small space. CA seems helpful in that software packages are available to build the system. However, we are particularly interested in more realistic sense of movement, rather than in tile-based simulation (arguably, if the tile size is small enough), and also in the way that people perceive things in an environment in the term of **objects** not individual cells. As a result, we believe an individual-based model using **spatially explicit mobile agents in continuous space** appear to be more appropriate for our approach.
3.3 Object-Oriented Programming

Everywhere we look we see objects, people, animals, plants, cars, planes, buildings and so on. People have an ability of abstraction, which enable us to view screen images as objects such as people, trees, and mountains rather than as individual dots of color. We can also, if we wish, think in terms of beaches rather than grains of sand, forests rather than trees, and houses rather than bricks.

We can divide objects into two categories – animate objects and inanimate objects.Animate objects are alive in some sense. They move around and do things such as people or animals. Inanimate objects such as buildings seem not to do much at all; they just sit around. All these animate and inanimate objects, however, do have something in common. They all have attributes like size, shape, color, weight and so on. And they all exhibit behaviors; for example, a pedestrian walks, stops, eats, sits and talks while a wall stands, obstructs a walk and blocks a view. Humans learn about objects by studying their attributes and observing their behaviors. Different objects can have similar attributes and can exhibit similar behaviors. Comparison can be made for example between children, middle-aged, and elderly people. A fountain, a fun sculpture, and an ice cream cart also have much in common.

Object-oriented programming (OOP) languages, such as Java, can be used to model real world objects with software counterparts. It takes advantage of class relationships where objects of a certain class, such as class of urban attractions, have the same characteristics. OOP gives us a more natural and intuitive way to view the programming process, namely by modeling real-world objects, their attributes and their behaviors. It can also model communication between objects, just as people send messages to one another. OOP encapsulates data (attributes) and methods (behaviors) into packages called objects. The data and methods of an object are thus intimately tied together. In C language and other procedural programming languages, programming tends to be action-oriented, whereas in Java Programming tends to be object-oriented. In C the unit of programming is the function while in Java the unit of programming is the class from which objects are eventually instantiated. Each Java class contains methods (behavior) corresponding to functions in C (Deitel, 1998).

By combining object-oriented Java programming with the ideas of multi-agent technology, we develop a model of how pedestrians interact in a 2-D experimental environment. A pedestrian is
represented by an agent-mouse, and the individual-based model is used to simulate the agent's behavior in the system.

3.4 System Architecture

We have created the simulation in the Java programming language, so the system can be displayed on a Java Applet, a multi-platform program that runs in a web page on any type of computer operating system with a Java enabled web browser.

The concept of the system design is derived from the intuitive way we view and understand the real environment. In a case, for instance, that we observe pedestrian movement in any given space, space organization – the interval, distances, and relationships between people and people, people and things, things and things – is apparently central in understanding the built environments (Rapoport, 1982). However, people live in time as well as space --the environment is also temporal, and can, therefore, be seen as the organization of time reflecting and influencing behavior in time.
The system consists of several objects organized in a hierarchical manner as shown in Figure 3.9. Basically, the structure of the system can be seen as a set of three consecutive levels. The first level contains a Java applet, which receives input from the user and displays output in a web browser. It also plays an important role as the constructor of all the objects for the simulation as will be described later. In the second level, there are three objects:

- House object,
- Time object, and
- Pattern object.

Whereas “House” plays a role as the environment, “Time” provides the time parameter for the simulation, and “Pattern” displays the visualization of movement patterns traced by the system. The Time object receives the user’s input through the applet’s graphical user interfaces (GUI). Finally, the third level, which contains four types of objects, extend from the “House”, indicating that “House” (House.class) accommodates all elements in an (urban) environment, including Mouse objects as pedestrians, Block objects as buildings or geometry in space, Cheese objects as urban attractions, and Gate objects as entry ways or exits to a given environment. Differing from other classes, Mouse.class is the only one in this system whose instances are animate agents, which have special abilities to move around in the virtual environment and interact with the other components. With the utilization of multi-agent technology, each autonomous agent “Mouse” employs simple individual behavior rules to perform its role as a pedestrian. The details of mouse behaviors in the simulation are discussed in the next chapter.

The mechanism of the system is based on the interactions among objects constructed by the applet, as seen in Figure 3.10. We can divide the system into four phases: Initialization, Creating, Simulation, and Ending. When the applet is launched in a web browser, it automatically builds three basic objects of three classes, House.class, Time.class, and Pattern.class, and stores them. Then it displays the House object, which is a blank white space by default, along with buttons, dialogs and menus to create and interact with the simulation. This is called the Initialization Phase of the system. As mentioned, a House object represents the environment, meaning that it contains all the “physical” elements that are important for creating a simulation scene. The elements are blocks, cheese, gates, and mice. In order to generate a simulation scene, the user must select and add elements to the scene and define all parameters necessary for the simulation. Note that in one simulation scene, there is only one House object, but there can be as many elements in the scene the computer memory can handle.
Figure 3.10: Construction of the system
The **Creating Phase** begins when the user starts to interact with the applet. A Block object is created when the user presses the “BLOCK” button and starts to draw a rectangle on the white space (House). Then, the user specifies the block’s character in the form of parameters (see Chapter 4). Not only the character, but also other parameters such as the location and size are also embedded in the object. These parameters will be maintained with the object until the object is either edited (ie., moved, resized) or deleted. After a Block object is constructed, the applet stores it in the House object. Each Block object has its own character, suggesting different interaction to the agent mouse.

A Cheese object is also created and stored in the House object in the same way as that of the Block object with its specific character (see Chapter 4). However, a cheese differs from a block in that it appears to be an attractor or stimulus, which can have a strong impact on mouse behavior at the motivated level.

Another type of element in the simulated environment is the Gate object. The applet reads the locations and sizes of all the Block objects stored in the House object, then it determines the locations for each gate. Gates are parts of the edges of the scene which are not occupied by any block. Finally, the applet constructs all the Gate objects and again stores them in the House object.

So far we have blocks, cheeses, and gates, which are considered as inanimate objects. They can have their own individual characters and certain behaviors, though they do not move around or interact with other objects. They just sit where they are as parts of the space configuration.

The last element to be instantiated before the simulation can take place is the Mouse object – the instance of Mouse.class, which is the main focus of this study. A Mouse object represents a pedestrian. Therefore, a simulation scene usually contains a large number of Mouse objects. A Mouse in this experiment is an autonomous agent because it is a component in the system that can interact with other components. An agent Mouse has its own internal structure (detailed in Chapter 5.1) which allows an ability to see objects in range, and its own “physical” mobility that enables its to move about and act upon the environment. Moreover, its behavior can be adapted and changed according to internal needs, external stimuli, and social interactions with other mice. The construction process of a Mouse object is more complicated than that of inanimate objects.
In this experiment, there are two basic types of mice: the Pink and the Blue, which represent the aimless wanderer and the purposive walker respectively. The user must input the total number of mice in the vicinity of the simulation scene and indicate the percentage for each type of mouse. Then, the applet constructs every Mouse object, gives it specific parameters, and stores it in the House object. The parameters are:

- the percentage of each type of mouse (pink and blue mice),
- the mouse’ s identification number (ID),
- the Time object,
- the Pattern object, and
- the House object.

The percentage of mouse types is used for a technical purpose. Basically, the mice that are outside the simulation scene are in the neutral state (neither Pink nor Blue). As soon as a mouse enters the simulation scene, it randomly changes to be a Pink or a Blue, with a probability dictated by the percentage numbers.

The mouse ID is used to identify each mouse in the system searching loops.

The Time object indicates a certain time of day that affects the internal state of the mouse. The internal states (or signals) such as the degree of hunger and the degree of relax-rush are initiated by the Time object. For example, if the Time object indicates that the simulation time is “noon”, the degree of hunger is likely to be at a higher degree than that of “morning” or “afternoon”. This also has an influence on mouse walking speed.

The Pattern object is given to every mouse so that each mouse can be able to leave its trace on the “canvas”, which can be displayed later by the Pattern object.

The last and probably the most important parameter that must be assigned to the Mouse object is the House object. The House object is first applied to every Mouse object to tell the mouse which environment it is in, and then each Mouse object is stored in the House object—the same House object. This technique enables each mouse to perceive all the elements in the virtual environment (House object) including itself and the other mice; we are in the environment and at the same time, the environment is in us in form of external perception.
As the Simulation Phase begins (when the user presses the “RUN” button), blocks, cheeses, and mice are drawn in the simulated scene by the House object. The applet displays the scene in the web browser’s screen. The House updates the scene. And, the applet re-displays the scene over and over again. This process generates a dynamic scene of mice moving in the virtual environment. While moving, the mice leave their traces on the “canvas” in the Pattern object in forms of pixel coordinates. At last, the user can see the mouse movement pattern by calling the Pattern object method to display the “canvas”. And, that would be called the Ending Phase.