UNIVERSITY OF OXFORD BROOKES

FINAL YEAR PROJECT

Simulation of the pedestrian flow in the context of fire evacuation in buildings

Author:
Guillaume ARDAUD
(09010297)

Supervisor:
Dr. Hong ZHU

Department of Computer Science - Module U08096

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Professor Hong Zhu for his supervision, advice, support and insight during this project.

I would also like to thank the Oxford Brookes professors I had this year, especially Dr. Clare Martin and Dr. Fabio Cuzzolin for their kindness, knowledge and support.

Finally, my special thanks to Dan Frost and David Lightfoot for nurturing the partnership between Oxford Brookes and the IUT2.

Contents

1 Introduction **7**

1.1 Motivation 8

1.2 Related Work 10

1.2.1 buildingEXODUS 12

1.2.2 SimWalk 13

1.3 Overview of the Project 15

1.3.1 Editing the building 15

1.3.2 Running the simulation 16

1.3.3 Gathering the results 17

2 The Proposed Approach **18**

2.1 The Main Functions of the System 19

2.2 The Architectural Design 20

2.3 Discussions 21

3 Design and implementation **22**

3.1 Environment 24

3.1.1 The environment 24

3.1.2 The actual design of the building 26

3.2 Simulation of Fire 29
3.2.1 The propagation of the fire 29
3.3 Simulation of Human Behavior in the Environment 32
3.3.1 Panic behavior 32
3.3.2 Optimal behavior 33
3.3.3 Follower behavior 33
3.3.4 Choice of behaviors 34
3.3.5 The Behavior Class 35
3.4 Discussions 37

4 Evaluation and Case Study **38**

4.1 Case Study 1 39
4.2 Case Study 2 41
4.3 Evaluation 43

5 Conclusion **45**

5.1 Summary of the Main Results of the Project 46
5.2 Comparison with Existing Work 47
5.3 Future Work 48
5.3.1 Improving the overall UI and the simulation workflow 48

5.3.2 Improving the accuracy of the fire simulation 50

5.3.3 Improving the AI of the agents 51

6 References **53**

List of Figures

1 Screenshot of buildingExodus in action, simulating the evacuation of a nightclub 12

2 Screenshot of Simwalk in action, simulating the evacuation of a casino. 13

3 Screenshot of Kitsune's toolbar. The tools allow the placement of, respectively, a wall, fire, an agent, the removal of an agent, a door, a fire exit, a staircase a label, the removal of a label, an additional floor, the removal of the topmost floor, and the starting of the simulation. 15

4 Screenshot of Kitsune's editor pane, depicting a single floor building with several agents and a source of fire. 16

5 Screenshot of Kitsune's inspector pane. 17

6 Screenshot of the log file created by Kitsune. 18

7 The Kitsune logo 20

8 The Test Driven Development Approach 23

9 UML class diagrams for the environment 25

10 UML class diagram for the undirectedGraph class 27

11 Step by step evolution of the fire simulation on a 3-by-3 grid 31

12 UML class diagrams for the Agent class, as well as the Behavior class and its daughters. 36

13 The layout for the first case study. 39

14 The layout after the simulation has been run for the first case study. 40

15 The layout for the second case study. 41

16 The layout for the second case study at the end of the simulation. 43

17 Application workflow 49

1 Introduction

Every year, fires in buildings are the cause of many deaths and injuries. In the United States, during the year of 2005, there have been 3,055 deaths and 13,825 injuries caused by fire in residential structures alone [1]. Even though it is impossible to foresee every accident, prevention plays a major role in saving lives.

There are numerous factors that can be influenced: training people to evacuate a site efficiently, building the constructions with less flammable materials, etc.

One parameter that is interesting to study is the pedestrian flow in the context of the evacuation of a building. The study of pedestrian flow is rather recent, so most buildings weren't studied as to achieve maximum evacuation in that respect.

As my final year project at Oxford Brookes University, I worked on the simulation of pedestrian flow in the context of fire evacuation in buildings through the building of a tool, named Kitsune, that would allow to design a building, place fire sources and pedestrians, run a simulation and gather the resulting metrics.

This final report details the approach that was chosen in the designing of such a tool, and then the design and implementation choices made. It then proceeds to cover the functionning of the algorithms and data structures that were used. We then continue with case studies of the software as it could be used in a real life situation, before finally discussing the outcomes of the projects, compare it with existing solutions, and consider what could be added or improved in the future.

1.1 Motivation

While it could seem trivial on the surface, the design and implementation of a tool that simulates the evacuation of a building on fire covers a wide variety of topics in computer science.

When deciding upon a final year project, my main motivation was to work on a subject that had to do with Artificial Intelligence, as I wish to pursue my studies in this field.

However, AI is a very broad field, and can hardly be treated in itself. The two subfields that interest me the most are swarm intelligence and genetic programming. After talking with several teachers of the computer science department about my interests, it appeared that Dr. Zhu's work was heavily related to swarm intelligence and emergent behaviors.

Even though I was interested in these fields, I had to decide to work on a concrete application, as the purpose of this final year project is not to do any original research, but rather apply knowledge to build the solution to a problem.

Furthermore, discussing with Dr. Zhu, we realized that safety is a growing concern in many aspects of our daily lives. Whether it is in automobiles, hospitals, offices, homes, the safety of the human beings involved is a top priority. In this era, our computational possibilites grow by the day, and it is most constructive to apply them to these needs.

To this extent, we came to the conclusion that artifical intelligence is a great mean to simulate real life situations in order to achieve this goal of maximum safety without any "trial and error" that would imply human lives.

Fire hazards are one of the most important safety preoccupations due to their destructive, costly and highly unpredictable nature. I thought it would be interesting to simulate the evacuation of a single building- and expanding on that I decided to buil a generic tool that would allow the modelisation of a building with various corresponding parameters, and then a run the simulation of a fire outbreak.

The agent simulation part is relative to the field of artificial intelligence and more specifically swarm behavior. Swarm behavior, inspired by natural phenomenons such as the flocking of birds or the schooling of fish, is relevant to multi-agent simulation- its aim is to make use of a context in which the interactions between the agents is key to the resolution of the given problem, as opposed to single agent applications in which only one agent solves a given problem.

Another aspect of artificial intelligence this project covers is pathfinding. This topic is used for the behavior of some agents, that have the knowledge of an optimal path to the fire exit.

Finally, the project also covers the implementation of algorithms to interesting means; for instance the simulation of fire using a cellular automaton, as we will see in section 3.2.

As a result, the motivation for this project is dual: on one hand it is interesting from a practical perspective, for it deals with real world issues that have a direct and concrete outcome; and on the other hand it is relevant from a research point of view, for the breadth and complexity of the theoretical topics it spans.

1.2 Related Work

The topic of the project belongs to the branch of artificial intelligence that is multi-agent simulation. Multi-agent simulation, as opposed to single-agent simulation, refers to applications where intelligent agents interact with one another to solve a problem, instead of the problem being solved by one single entity. Multi-agent systems are often used for complex problems who could not be realistically defined and solved using a single agent system.

Agents in a multi-agent system have certain characteristics [2]:

- They are autonomous- which means they do not require any external help to function
- They only have a local view of the system
- There is no controlling agent

An interesting property of multi-agent systems is the notion of emergent behavior. An emergent behavior is a behavior that arises from the interaction between several agents, and that was not envisioned during the conception of the system. Emergent behaviors are often found in nature- for example, when birds flock or ants build an anthill. Emergent behaviors are a strength of multi-agent systems, for they bring new insight on how to solve a given problem- insight which would not have appeared if a single-agent system were used. [3]

With the evergrowing preoccupation that is safety in our modern day society, multi-agent simulation has naturally evolved to simulate processes which study can then be used to learn about them. In our case, the process is pedestrian flow, or how people move and organize themselves in specific contexts. The topic of pedestrian flow simulation in the context

of an evacuation is a deep one, and many research facilities throughout the world study it (an annual conference is dedicated to it, PED (Pedestrian Evacuation Dynamics) [4]). Whether it is because of natural disasters (earthquakes, storms, fires,...) or human ones (terrorist attacks, bombings,...), evacuation is a much studied subject, and any possible way to improve building layouts and evacuation techniques results in saved human lives. As a result, many papers describing pedestrian simulation algorithms have been published in the past few years. [5] [6] [7] [8]

Cellular automata have been used extensively for the modelisation of the fire in an accurate manner, albeit limited to the 2-dimensional plan [9]. This has led to the choice of such technique for our purposes.

However, due to time restrictions and the fact that I could not cover in less than 6 months of undergraduate personal research the amount of material that has been developed for years by entire research teams that are devoted full time to it, these papers and algorithms could not be implemented and compared extensively in the project. Instead, I have inspired myself from the techniques presented, implementing them in a way that I thought was efficient, simple, and flexible.

Below is a short presentation of the main tools that are available for simulating pedestrian flow in the context of building evacuation. Unfortunately, as these tools are commercial solutions, I could not obtain a copy of them for testing. The observations made are hence derived from the projects' webpages and FAQs, as well as some demonstration videos I could find on sites such as YouTube.

Please note that in this section, we only present the main software that already exists. For a more complete parallel between these software and the work that has been done on Kitsune, see *5.2. Comparison with Existing Work*.

1.2.1 buildingEXODUS

buildingEXODUS is a solution developed by the Fire Safety Engineering Group (FSEG) from the School of Computing and Mathematical Sciences at the University of Greenwich [10]. It is part of the larger EXODUS family, designed to simulate evacuation of people in various environments (the family includes railEXODUS for trains and railways, airEXODUS for the aviation industry, etc.).

buildingEXODUS is however fairly basic in its simulation of behaviors- it is impossible to specify individual behaviors for the agents, and they all share a common, basic one: get to the nearest fire exit as fast as possible.

Figure 1: Screenshot of buildingExodus in action, simulating the evacuation of a nightclub

It should be noted that in itself, buildingEXODUS just simulates the pedestrian evacuation- not the spreading of the fire, for instance. For simulating specifically fire evacuations, buildingEXODUS can be used in con-

junction with SMARTFIRE, developed by the FSEG as well.

1.2.2 SimWalk

SimWalk is a generic pedestrian simulation software developed by Savannah Simulations [11]. Its applications range from building evacuation to public transport simulation or urban planning.

Figure 2: Screenshot of Simwalk in action, simulating the evacuation of a casino.

The algorithms it uses for pedestrian simulation are very thorough, borrowing equations from granular physics for instance. As a result, it allows the simulation of a very large number of pedestrians, as can be seen on figure 2, with a great level of realism. Unfortunately, like buildingEXODUS, it is impossible to specify complex, per-agent behaviors.

Similarly to buildingEXODUS, SimWalk does not simulate fire, merely the pedestrian evacuation- to be used to this effect, it must be used in conjunction with other software.

1.3 Overview of the Project

Kitsune is fire evacuation simulation tool, implemented in Java and using the Swing framework. It allows the design of a building spanning several floors, the placement of agents in the building, and the simulation of a fire outbreak in said building. Once the simulation is over, it provides various metrics to evaluate the evacuation.

When launched, Kitsune consists of a single window, from which all operations are made.

1.3.1 Editing the building

The first step of Kitsune's workflow is editing the building's layout. This work is done using the mouse to select an icon from the toolbar (see figure 3), and then clicking in Kitsune's editor pane to edit the actual layout of the building (see figure 4).

Figure 3: Screenshot of Kitsune's toolbar. The tools allow the placement of, respectively, a wall, fire, an agent, the removal of an agent, a door, a fire exit, a staircase a label, the removal of a label, an additional floor, the removal of the topmost floor, and the starting of the simulation.

To edit the building, the user must first create a floor in the building. After that, using the mouse and by selecting the appropriate icon in the toolbar, he can place walls (represented in black), doors (represented in green) and fire exits (represented in pink) on the grid. By clicking on a component a second time, it is erased. The user can also set the intensity of the fire on each cell of the grid- clicking on a cell while having the fire tool selected cycles the cell to the next level of fire intensity. Finally, the

user can also place or remove agents, and labels (these have merely an esthetic function, allowing the naming of rooms for instance, making the map more readable) as well as remove the topmost floor from the building.

Unfortunately, due to time reasons, the main feature that was not implemented are staircases and the integration of multiple floors in the simulation.

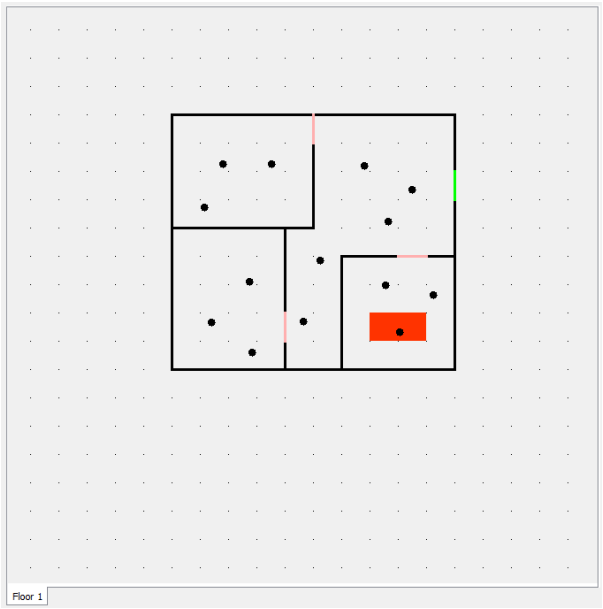

Figure 4: Screenshot of Kitsune's editor pane, depicting a single floor building with several agents and a source of fire.

1.3.2 Running the simulation

When the editing is done, the second step is running the simulation itself. This is done by clicking the "Play" button in the toolbar. When clicked, the simulation is launched and the button becomes a "Pause" button. At any time during the simulation, this button can be clicked to pause the simulation, if the user wishes to inspect the building's state at this point in

details, for example. Clicking the button again will resume the simulation.

The simulation goes on- even if all the agents have exited the building, it pursues the simulation of the fire- until the user stops it manually by pressing the pause button.

1.3.3 Gathering the results

Finally, the results are displayed in the right-hand pane (see figure 5). They can be either read and exploited as is, or exported in a text file by selecting "Export Simulator Log" from the "Project" menu, which will create a file named "Kitsune log.txt" with all the information from the pane aforementionned (see figure 6).

Item	Value
Number of exits	1
Total number of doors	0
Total runtime (seconds)	18
Total number of agents	8
Total number of agents that escaped	0
Total number of agents that are exiting	8
Total number of agents that are stuck	0
Average time for exit	0
Time of first exit	0
Time of last exit	0
Success ratio	

Figure 5: Screenshot of Kitsune's inspector pane.

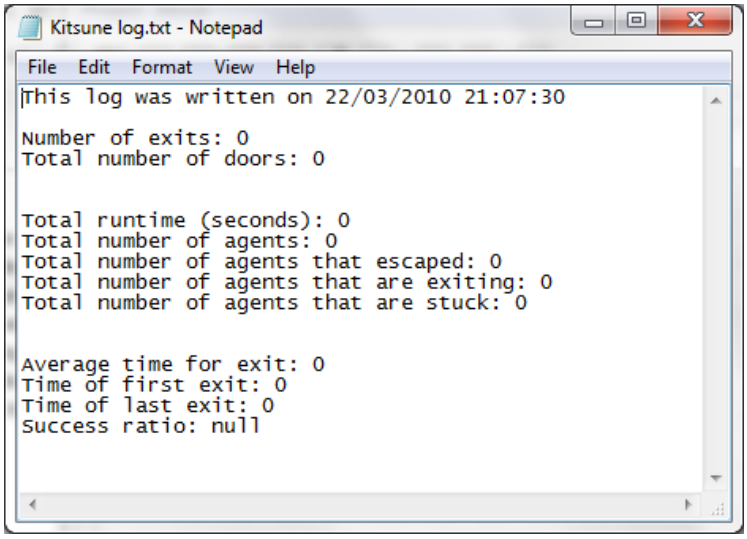

Figure 6: Screenshot of the log file created by Kitsune.

2 The Proposed Approach

The proposed approach for this undergraduate personnal project is to design a software from the ground up that will allow the simulation and study of the pedestrian flow during the evacuation of a building on a fire.

This software will be auto-sufficient- it will not require any additional executables or libraries installed in order to run it, and all the operations will be realized within it.

2.1 The Main Functions of the System

The system itself has 4 main functions:

1. Allow the **design and creation of the environment** in which the simulation is going to take place.
2. **Simulate the spreading of the fire** in the environment, in a manner as efficient as possible.
3. **Simulate the pedestrian flow** resulting from the evacuation of the environment on fire, in a manner as realistic as possible.
4. **Provide** the user with **various metrics**, once the simulation has ended, in order to analyze how the simulation went.

These four main functions of the system are all bound very tightly together, as they are all part of the same coherent application, and aim to provide a seamless workflow.

2.2 The Architectural Design

The system will be implemented in a desktop graphical application, Kitsune, that will allow the use of its functionnalities in a intutitive and streamlined manner.

In order to achieve this, we will be using Java programming language, a modern, object-oriented language that is used a lot in the corporate context, and increasingly so in research, especially artificial intelligence. This success is due to its modern features requiring less low-level work from the programmer, allowing the focus to be on the actual implementation of algorithms. The language is also not directly compiled in machine code, but rather in bytecode, allowing its execution on virtually any modern platform, with a speed comparable to compiled code.

In addition to Java, we will use the Swing framework, a native Java library, studied for the development of graphical desktop applications. Swing provides all the traditional components used in graphical applications (panels, tabs, buttons, sliders,...), and implements the MVC design pattern, which separates the presentation from the content and the operations, thus making the application's architecture even more flexible.

Figure 7: The Kitsune logo

2.3 Discussions

This approach is ambitious, in that it covers a variety of topics, some of them barely related to the title of the project (for instance, the creation of a tool that allows the design of the building is not related to the study of pedestrian flow in itself).

However, due to the lack of open source basis available, and to the fact that all parts of the system need to be coherent and fit together, we will have to focus on the development of all of them at the same time. Naturally the process will be much more time consuming than if we had focused on just one function of the system, but we believe that in the long term this approach will be worthwhile for its thoroughness.

In that respect, even though the project may not reach all of its initial goals, it will provide a solid foundation for any project wishing to build upon it in the future.

3 Design and implementation

Kitsune was implemented using the Java language. The language was chosen for its flexibility as well as its efficient implementation of the object-oriented paradigm. It offers many high level functionnalities such as garbage collection, which reduces the amount of time spent optimizing and debugging the codebase.

The Java library used for the drawing of windows and the visualization of the simulation was Swing. Swing is a highly popular library in enterprise and research contexts alike, due to the variety of the components it offers. Furthermore, Swing implements the model-view-controller (MVC) paradigm very elegantly, making the implementation process more transparent, debugging easier as well as the code cleaner all the way through.

The design was made using the Unified Modelling Language (UML), specifically class diagrams. Due to the low footprint of the application, use case diagrams and others such as sequence diagrams or state machine diagrams weren't made. A direct result of using UML for design is that once the design is complete, the implementation is very straightforward, as class diagrams and the structure of object oriented programs are nearly identical- hence the whole structure of the components is already coined down when the actual programming starts. The final design, however, does differ slightly from the initial one, as development always reveals shortcomings and flaws in the original models.

Finally, the implementation was approached in the traditional iterative manner, also called Test Driven Development (TDD). In TDD, the idea is to always have a build of the code that runs, and then progressively add features to it, while debugging said features (see figure 8). As a result, debugging times are greatly reduced, and it is easier to evaluate the current state of the project due to the fact that there is always a running, stable binary available.

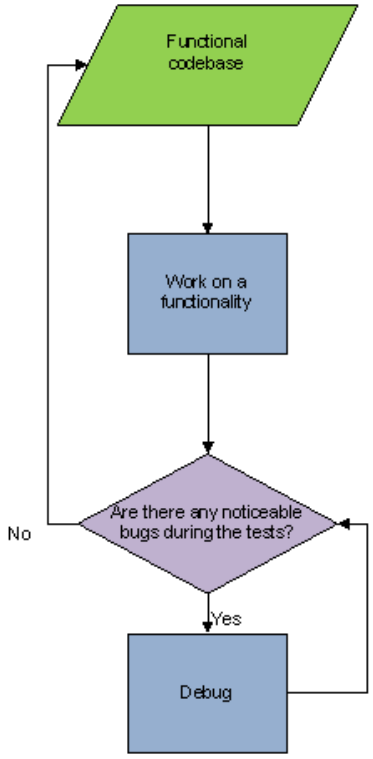

Figure 8: The Test Driven Development Approach

3.1 Environment

The environment portion of Kitsune is the basis on top of which the fire and the agents will evolve; as a result it must be flexible, and use solid data types that are easy to manipulate. This section describes the architectural design of Kitsune's environment, from the ground up.

3.1.1 The environment

This section describes the data structures used behind the environment in Kitsune. For all discussion of classes implementation, refer to figure 9, which is the UML diagram for the environment classes.

The environment is displayed in a Java Swing JPanel, which allows direct bitmap drawing. As a result, the main class, **buildingFloor**, is a daughter class of Swing's JPanel. This enables the class to have its own method, *paintComponent()*, to draw the visualization of the simulation, layer by layer (ie. first the grid, then the fire, then the walls, doors, exits, then the agents, etc.).

The **buildingFloor** class holds a single **Environment** object, which contains all the data structures relevant to the current environment. As we can see on figure 9, the **Environmen**t object has an **ArrayList** of agents, which contains all the agents in the current environment, as well as an **ArrayList** of **Level**, where one **Level** represents one floor in the current building. The **Level** object holds 3 **UndirectedGraph**, one for the doors, one for the exits and one for the walls, as well as a **Fire** object which in it turns holds a matrix representation of the fire on the current level. As it is standard in the object oriented paradigm, each object contains the methods relevant to the manipulation of its data structures.

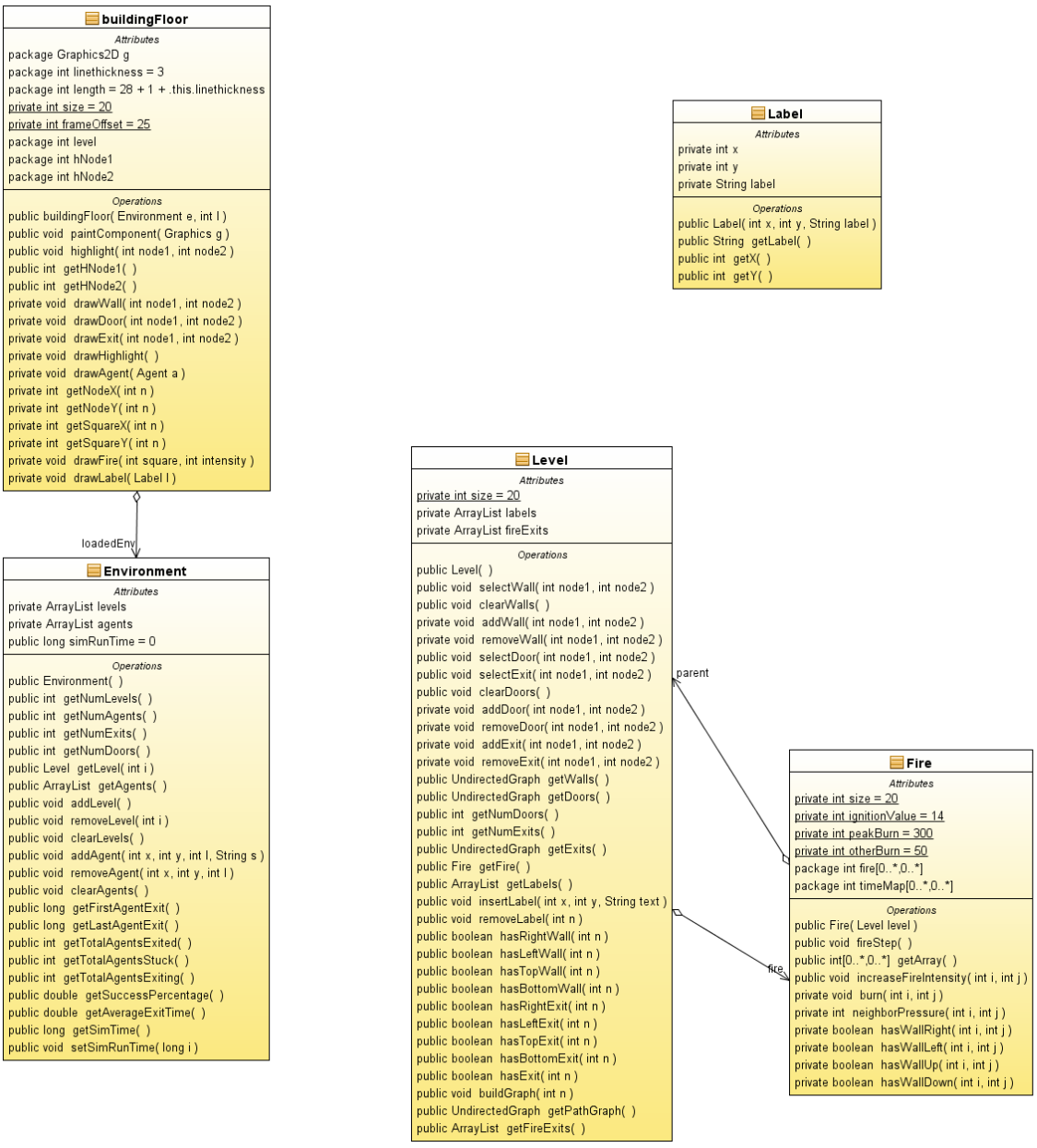

Figure 9: UML class diagrams for the environment

3.1.2 The actual design of the building

To represent the building layout, we use an undirected graph- each point of the drawing grid is a node of the graph, and each wall is an edge between two walls. The environment has several graph representations stored in memory: on for the walls, one for the fire exits and one for the doors.

There are several ways of representing an undirected graph- the main ones being the adjacency list, and the adjacency matrix.

Both approaches have their advantages and inconvenients. Adjacency matrices store information about the status of every node and edge, whereas adjacency lists only store information about the nodes that share edges with other nodes. As a result, the matrix approach allows faster operation, as looking up for the existence of an edge is merely a table look-up, and therefore has an algorithmic complexity of *O(1)*. In an adjacency list however, there is a worst-case algorithmic complexity of *O(n)*. It is therefore a question of whether processing power or memory should be preserved.

In our case, the graph-space is rather big (one floor requires three matrices of n*n[1] nodes[2]- therefore in a multi-level building, we would quickly have several thousands of nodes. However, the graph is also very sparse, due to the nature of the building representation. If we were to consider a 3-floor building with 40 walls on each floor all connecting different nodes, using 8-bit integers, the total memory taken would be `3 floors * (20*20) matrix * 3 matrices =3600` octets in an adjacency matrix, and `3 floors * 80 edges described in the list * 3 lists= 720` octets in an adjacency list, a 500% difference.

The huge difference in memory occupation of the data structure, as

[1] In our implementation, n=20

[2] One for the nodes, one for the doors, one for the fire exits

well as the fairly low computation difference in our situation, makes the adjacency list an obvious choice.[3]

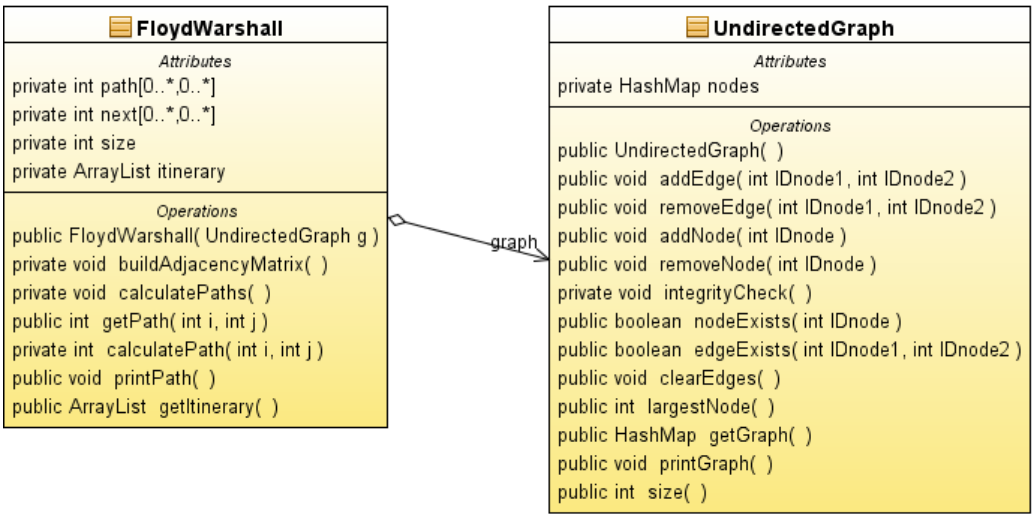

Figure 10: UML class diagram for the undirectedGraph class

The architecture of the **undirectedGraph** class itself is very straightforward, including methods for common operations on a graph (see figure 10 for the UML class diagram of the undirected graph). As a result, any graph algorithm (Dijkstra, Floyd-Warshall, Ford-Bellman, etc.) can easily

[3]Even though the memory usage difference is huge, on modern machines with several hundred or thousands of MB of RAM, no significant difference would be observed between the two approaches however.

be implemented- which is exactly what we do when implementing agent behavior classes (see 3.3.).

As a matter of fact, for the purpose of the follower and optimal behaviors, we implemented our own class that uses the Floyd-Warshall algorithm [12] to calculate the shortest path in the given graph between any two nodes.

3.2 Simulation of Fire

The fire is a central part of the simulation, therefore a lot of time was spent researching the existing algorithms or simulation of fire, testing various approaches and techniques.

3.2.1 The propagation of the fire

The environment is represented by a m*n sized grid (in our implementation, n=m=20). Each cell of the grid has two values that are attached to it- the intensity of the fire burning on the cell (represented by the fire map) and the number of steps during which the fire has been at its current intensity (represented by the time map).

The fire can take 8 different values for its intensity: either burnt or unburnt, or one of six burning intensity.

The rules used for the propagation of the fire are inspired by cellular automaton. However, classical cellular automaton only reason only using spatial dimensions- due to the nature of fire, we use a temporal dimension as well.

In the current state of the implementation, we update the time map and the fire map every step (in the current state of the implementation, there is one step every 100 ms) using the following rules:

- If a cell is burnt, then it will never change state again
- For each cell in the grid, we calculate the "pressure" that is exerted onto it by its neighboring cells. The pressure is calculated by summing the intensity of all neighboring cells multiplied by a burn prop-

agation coefficient (the coefficient will be lower when a wall or a door is present for instance, as in real life these would slow the spreading of the fire).

- If a cell has been at the peak of burning (ie. the value of its intensity is 6) for more than 30 steps, then it becomes burnt.
- If a cell has been at its current burning intensity for more than 10 steps, then its intensity increases by one.
- If the pressure exerted on a cell exceeds 15, then the cell is ignited. It is interesting to note that burnt cells have a value of 1, and therefore can contribute to the ignition of a neighboring cell. This seems relevant as in real life, there would still be ashes and cinders in burnt areas that could potentially ignite nearby areas.

Figure 11 shows the evolution of a 3-by-3 portion of the grid that would be on fire, with the accompanying time map and fire map.

We can see that the central cell is ignited at time n+1, as the sum of the intensities surrounding it exceed the defined value (15). At time n+2, the 4 cells that are directly adjacent to the central cell see their intensity increased by 1, as they have been at their current intensity for the defined amount of steps (10 steps in the current implementation).

All in all, the resulting modelisation of the fire is not as accurate and detailed as it could be. However, it provides a solid starting point for the simulation, and still manages to provide a correct yet rough representation of the fire propagation. The algorithm is flexible (through the use of burn propagation coefficients) and light enough that it can easily be adapted to work in a much more detailed context in the future (for example by increasing the size of the grid), and that the fire propagation rules can be complexified.

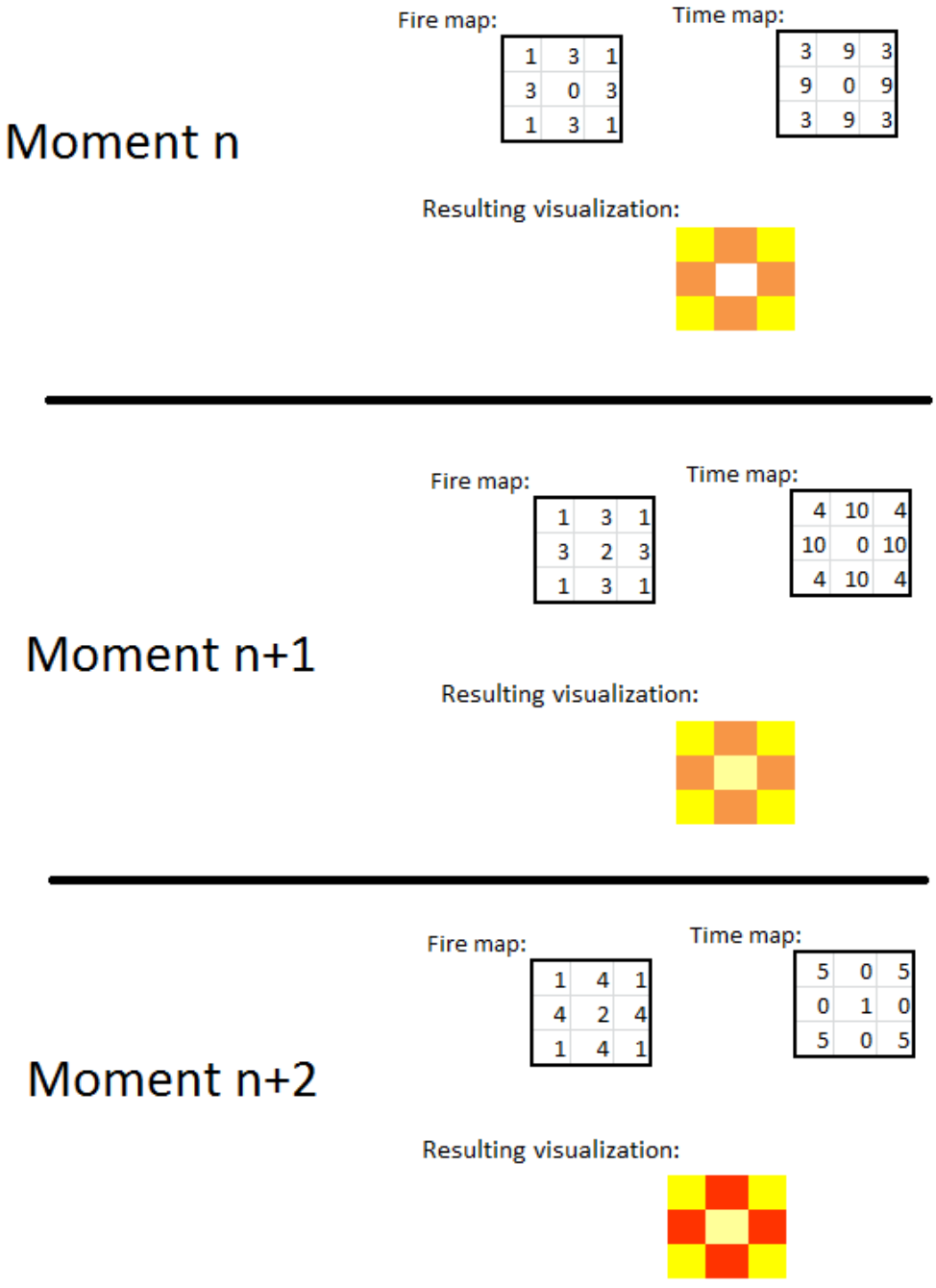

Figure 11: Step by step evolution of the fire simulation on a 3-by-3 grid

3.3 Simulation of Human Behavior in the Environment

In order to try to implement the agents in the most realistically way possible, several behaviors were devised for the agents. These behaviors are set at the creation of the agent in the building editing stage, and cannot be changed. Three basic behaviors were developed for Kitsune, that we will describe in their details.

Additionally, no matter what their behavior is, if an agent is on a cell on fire, than he takes damage from the fire- when a certain amount of damage is taken (predetermined by a *life* attribute to the **Agent** class that diminishes with damage taken), the agent dies- on the visualization, he becomes white and stops moving.

3.3.1 Panic behavior

This behavior is very straightforward, and aims to model an individual that is panicked by the fire or that would be confused by the situation (for example a toddler).

To achieve this, the agent builds an internal representation of the map, using a matrix where each cell of the matrix represents a cell on the building's grid. Whenever the agent is in a cell, he marks it as visited. He then looks around for cells immediately adjacent to his that he hasn't visited, and randomly picks one. If he is on a cell with a fire exit, then he takes the exit. If there is no available cell around him, the agent enters a stage where he is immobilized by panic, believing to have exhausted all of his possibilities, and remains immobile.

It could certainly be enhanced to make it more realistic (for example, the agent would make its decision based on the intensity of the flames in

certain areas, recover from immobilization after a short while, etc.). However, in the state of things, it is a good behavior to showcase the possibilities offered by Kitsune, and run standard simulations.

3.3.2 Optimal behavior

This behavior is optimal in that the individual takes the shortest route possible to the nearest fire exit. It aims to model individuals that went under fire evacuation training, for example, and have knowledge of the building's layout.

In order to achieve that, we proceed in two steps.

1. First, when the simulation starts, we generate a graph of the building, using each cell as a node, and connecting cells that are not separated by walls. We model the graph using an adjacency matrix in order to simplify the following step.
2. Then, using Floyd-Warshall's algorithm [12], we calculate the shortest route for each agent to the nearest fire exit.

The agent will then follow that path during the entire run of the simulation.

3.3.3 Follower behavior

This last behavior is perhaps the most straightforward to detail- the agent merely picks the closest agent, and follows the shortest path to him. The outcomes for the concerned agents are variable- it is highly effective when following an optimal behavior agent, and not very effective when following an agent that adopted a panic behavior. It is rendered utterly useless if

the targetted agent is a follower who follows the targetting agent, resulting in a deadlock.

This type of behavior is plausible in real life- it is highly common, during a catastrophe when a person is in a state of shock, to just follow the people around them, even if it does not result in the best possible choices.

3.3.4 Choice of behaviors

The reasons for which these three specific behaviors were chosen are varied.

First of all, they were designed to showcase the capabilities of Kitsune. As such, they were intended to be rather generic in their functionning, to demonstrate the system at large. Finding the optimal path in a graph is a classical problem in AI, and we thought it'd be interesting to apply it here. The follower behavior was inspired by BOIDS [13], an algorithm which aims to simulate the flocking of birds. In order to achieve this, each bird follows the average direction and speed vector of the others, resulting in an homogeneous group. Finally, the Panic behavior was inspired by a classical maze solving solution, which consists in always turning in the same direction. Here however, we thought it would be interesting to add a random component to the technique, so that the odds of getting to the fire exit would not be always in favor of the agent.

A second reason for choosing these behaviors is that they are pretty basic, and can therefore be used as "construction bricks" for more complex behaviors that would evolve dynamically if someone were to pursue the work on Kitsune.

Finally, as stated in the description of each behavior, we chose behav-

iors that were close to the ones we would typically observe in a fire evacuation: the one proper to a person who kows the evacuation route, the one proper to a person who is panicking because of the fire, etc.

3.3.5 The Behavior Class

In the application code, each instance of the Agent class has a reference to one of the behaviors, passed to the constructor at object creation. Each behavior has its class that inherits from the abstract Behavior parent class. These daughter classes implement their own version of the *action* method, which is called at each step in the simulator.

In this manner, additional behaviors can easily be added to Kitsune in future versions (see 5.3.), thus making the application more flexible and extending its possibilities. The steps to follow to add a new behavior to Kitsune are pretty straightforward:

- Create a new class that inherits from **Behavior** and implements the **Serializable** class. Its constructor should also define the agent's color for display, define the string that defines the behavior, and assign the parent to the class's parent attribute.
- Override the *step* and *calculateNewSquare* methods. The former is called at each step of the simulation, while the latter is used to attribute a new square towards which the agent will walk.
- Finally, modify the **behaviorPicker** class to allow the selection of the newly created behavior. It is recommended to use Netbeans for the edition of the **behaviorPicker** form, in order to maintain the integrity of the XML.

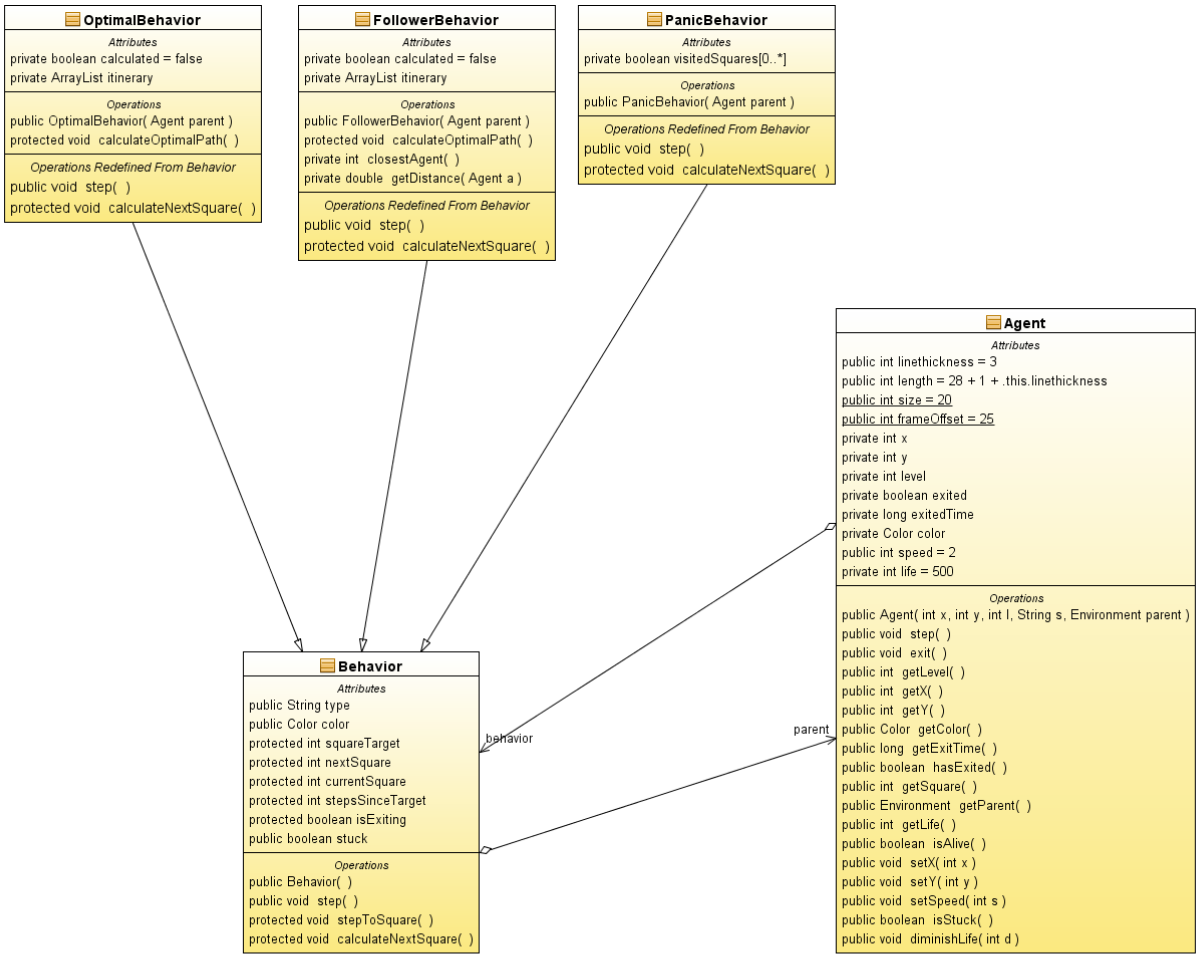

Figure 12: UML class diagrams for the Agent class, as well as the Behavior class and its daughters.

3.4 Discussions

There are still several considerations to bear in mind regarding the current design and implementation.

First, concerning the environment, the way the layout of the building was represented can be limitative if the application were to be heavily extended, to support for example curved walls, or a floor with different elevations, etc.. The use of bitmap masks - or even better, vector masks- for the representation of obstacles and objects seems more appropriate in this case- it would allow a greater flexibility in the building design, and would also allow per-pixel detection collision for the agents, the spreading of the fire, etc. instead of a reasoning on a per-cell basis.

Then, in regards to the fire, a key thing to keep in mind is that the use of a cellular automaton for the modelisation of the fire is only relevant because we model the far on a 2-dimensional plane (as the building is viewed from the top). If we wanted a simulation that matched the real world as closely as possible, a radically different approach would have to be envisioned.

Finally, regarding the design and implementation of the agents, the behaviors are far from perfect. The human mind is a complex thing, and there are many things that could be done and many aspects that could be dealt with to get a more accurate simulation. For instance, the behaviors could take into account the immediate risk that an agent is in (depending on the intensity of the fire surrounding it) and adapt the exit trajectory accordingly. Other ideas concerning what could be added to the agents are discussed in 5.3.

4 Evaluation and Case Study

In this last section before concluding, we will review two case studies of Kitsune.

These case studies have several objectives. First of all, they ensure that the software is useable under normal conditions- and while this does not prove it is bug free, it shows that it has a functionning minimum of stability.

Secondly, they allow to observe the results obtained in typical tests. These results allow to judge whether Kitsune is close enough to reality in its simulations, thus measuring its pertinence as a tool.

Finally, they also enable us to see which possibilities lack in the current implementation, giving us a better idea of what the priorities for future work should be.

4.1 Case Study 1

The first case study will use a very simple building layout, with only a few agents. The aim of this will be to verify that all agents follow the desired behavior, that the fire spreads properly, and that the application as a whole behaves properly.

The layout used as a starting point can be seen on figure 13. It can also be found on the CD in the *layouts/* folder, to be opened in Kitsune.

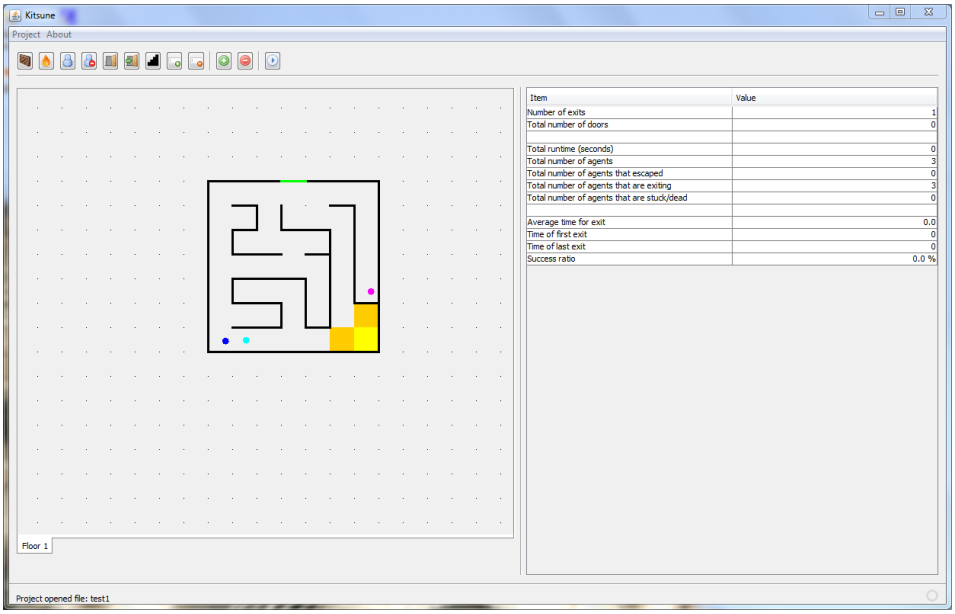

Figure 13: The layout for the first case study.

This layout is made of a single room with a fire exit located at the north. The room's layout, however, is quite convoluted so that we can observe the behavior of the agent with the optimal behavior and the follower. The agent in a state of panic is placed in a corridor, and therefore its chances of survival are quite high. The fire is placed away from the agents, and even though it will propagate it is unlikely anyone will get killed by the fire.

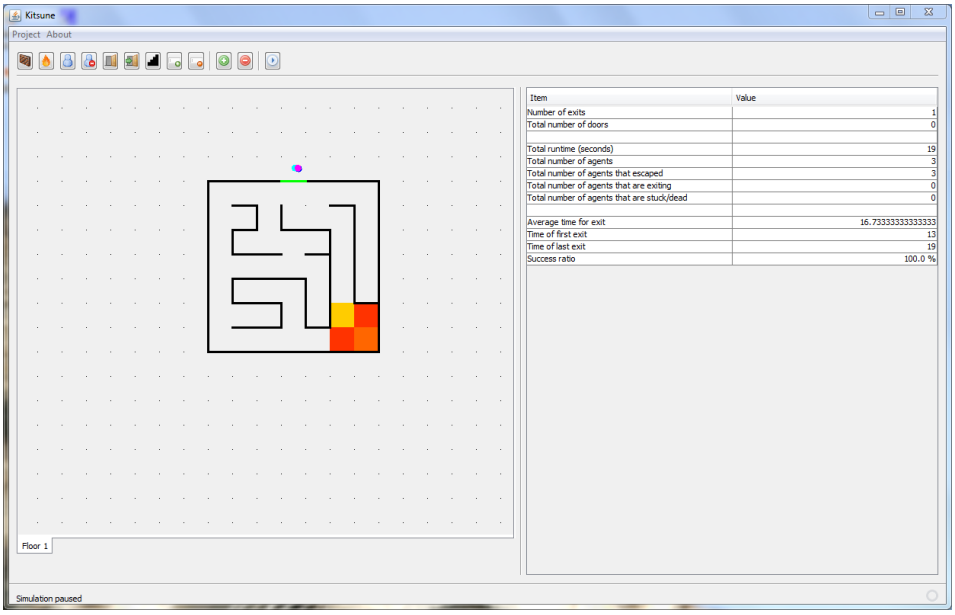

Figure 14: The layout after the simulation has been run for the first case study.

On figure 14 we can observe what the layout looks like once the simulation has been run. The agent with the optimal behavior has successfully exited the room using the least costly path, and has been followed by the agent set to the follower behavior. The third agent has escaped has well, even though this is non-deterministic. Finally, we can see the fire has started spreading, but did not affect any of the agents.

This first case study shows that Kitsune is able to perform all of the basic tasks that we defined during the conception of the project. It is now time to run a second case study, this time more complex and closer to real life applications.

4.2 Case Study 2

This second case study aims to be much closer to what a potential real use of Kitsune could be. Therefore, we are going to simulate the evacuation of a house, where the fire starts in the kitchen. There are several optimal agents, that could be considered the owners of the house (who therefore know their way out), followers, that could be considered guests, and agents who are in a state of panic- these could be for example toddlers in the bedrooms or bathroom.

The layout used as a starting point can be seen on figure 15. It can also be found on the CD in the *layouts/* folder, to be opened in Kitsune, where there are two files: *housePopulated*, which is the exact layout used in this section, and *house*, which has just the walls and doors, and can be used for experimenting with the placement of agents and areas where the fire starts.

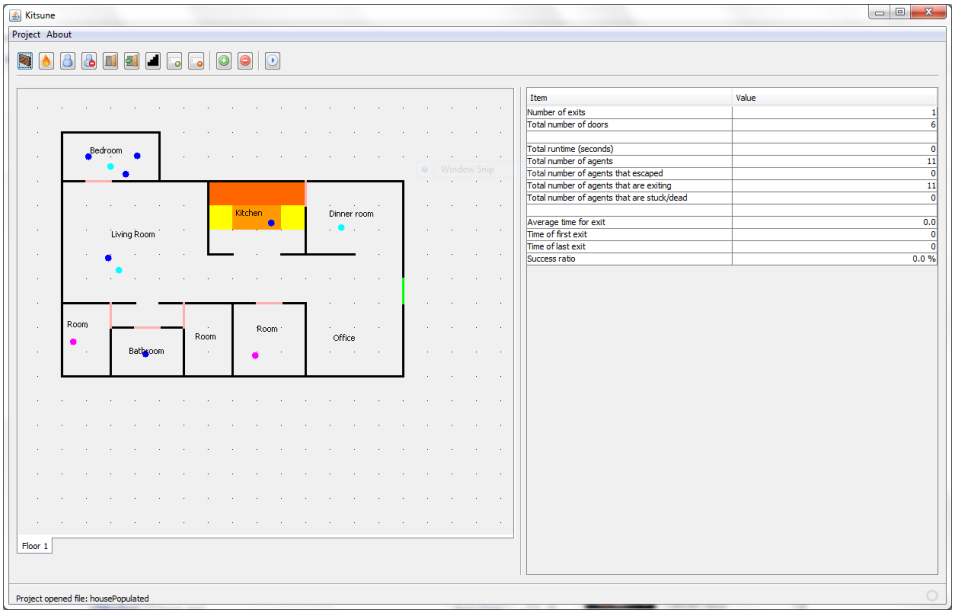

Figure 15: The layout for the second case study.

Once the simulation is over, we can export the log to examine it further.

```
   This log was written on 16/03/2010 10:14:19
Number of exits:  1
Total number of doors:  6
Total runtime (seconds):  39
Total number of agents:  11
Total number of agents that escaped:  9
Total number of agents that are exiting:  0
Total number of agents that are stuck/dead:  2
Average time for exit:  29.511777777777777
Time of first exit:  16
Time of last exit:  35
Success ratio:  81.81818181818183
```

This log tells us that 80% of the agents have survived- a number that is quite low, as every human counts and therefore our goal is a perfect evacuation each time.

We can see that the average time for exit is roughly 30 seconds, with the last exit being shortly after and the first one being much earlier. This tells us that there have been a lot of exits grouped near the 35 second mark, and only a few before then- which means that the fire exit was not conveniently placed for that last group of people.

While running a simulation gives us some insight on the layout of the building, we cannot draw empirical conclusions solely from that. To this effect, we would have to broaden the possibilities of Kitsune.

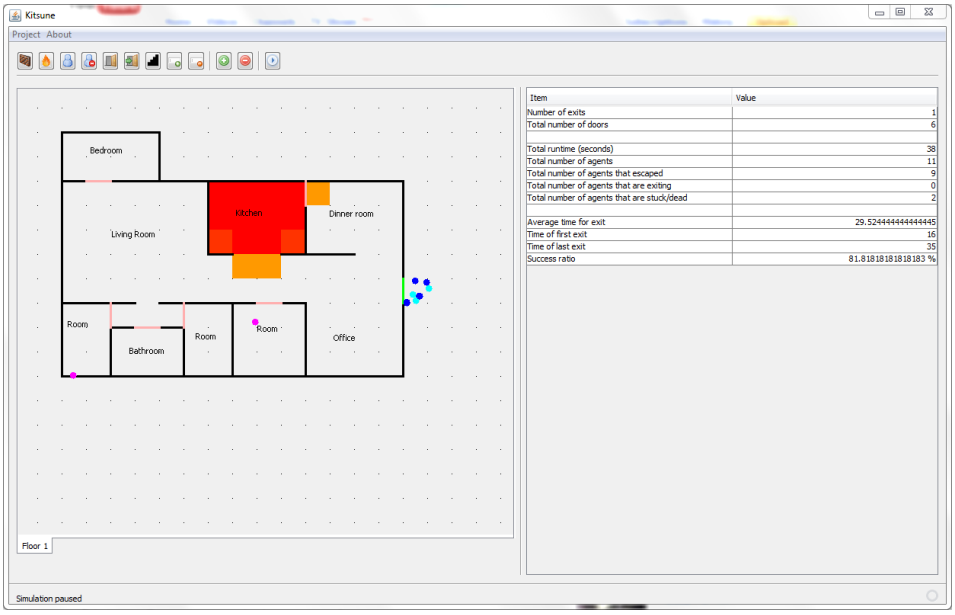

Figure 16: The layout for the second case study at the end of the simulation.

4.3 Evaluation

Looking at the case studies, it is clear that in real life, things would have been different- for instance in the second case study, the adults would have found the toddlers before leaving, for example. Therefore the 80% success ratio is flawed, and would have likely been 100% in an actual situation. It is errors like that that reduce Kitsune's utility as a simulation tool.

In addition to that, the testing process is not as fluid as it could be. One has to place the agents, sources of fire and fire exits manually on the building layout, which is repetitive and takes a lot of time. One could imagine a simulation mode where Kitsune generates automatically agents and fire patterns for a building, and can then draw statistics conclusions on the efficiency of fire evacuation in that building. Combined with deeper metrics, this would dramatically improve Kitsune's potential.

Overall, these case studies show us that while Kitsune is a solid start for the analysis of how a building is evacuated and how it can be improved in this respect, much can be done to make it a powerful tool. For instance, it is apparant that the integration of automated testing and the addition of more complex behaviors are two things that would greatly enhance Kitsune's usability and relevance.

5 Conclusion

In this report, we have covered the motivations and ideas behind Kitsune, as well as the architecture of the application as well as the implementations of the various algorithms and data structures used. We have also reviewed use cases for the application, outlining the uses and limitations of the final product.

However, as it was first hinted in the use cases Kitsune is far from complete. In this last section, we will first summarize the work that has been done, and review the experience and knowledge that was gathered during the project. We will then see how Kitsune compares with the existing software, and finally discuss the additional work that can be envisioned in the future.

5.1 Summary of the Main Results of the Project

Overall, this project was very formative for me. It allowed me to go through all stages of the development of a complete application, thus covering several areas of software engineering, computer science and mathematics alike. It was an engaging project to work on, both for its concrete goals and for the more theoretical concepts that lie behind it.

I think that a lot was accomplished. The whole environment for the simulator and visualization of the simulation is implemented, as well as rudimentary agent behaviors, including a mechanism that allows adding more in the future. Finally, a basic fire spreading model was put in place, that can easily be extended. Overall, all of these cover various topics that had to be studied and developed to work together, and I am satisfied with the work accomplished.

As stated in the various evaluations throughout the report, Kitsune is far from complete. This is due on one hand to the time factor, as it was a very ambitious project. It is also due to the fact that I spent a lot of time at the beginning of the project going through various articles and papers about multi-agent simulation, to try to wrap my head around what the project would require. In addition to that, the Swing API was completely new to me, and learning it and getting used to it took a bit of my time as well. However, I believe that it remains a solid application that can stand on its own.

5.2 Comparison with Existing Work

While Kitsune doesn't stand the comparison, as it is , it remains nonetheless a good exercise for evaluating the project and the work that has been done.

First of all, it is interesting to note that no tool offers a combination of fire simulation and pedestrian evacuation modelisation. As was pointed out earlier, tools are dedicated to pedestrian evacuation simulation, and can be coupled with fire simulation software if required. Even though that leads to more flexibility concerning the use of the software, it results in a reduction of the way fire and agents interact together. People being asphyxiated by smoke, being trapped by the fire, reacting to the dynamics of the fire, etc. are some of the many things that are not taken into account when using this "hybrid" approach to the problem.

Another noteworthy point is that these software use physic based approaches. These allow for larger and more scalable solutions, but however do not permit very accurate behavioral simulations. Humans are highly unpredictable in risky situations, and a software that uses a deterministic method of estimating human reaction will not be able to go beyond a certain inherently defined accuracy in its results.

When considering the amount of work put in these commercial solutions, it is normal to observe that they are far superior to Kitsune in realism, functionnality and relevance of the results that they output- in this respect, Kitsune is more of a proof of concept. However, it is very interesting to note that the approach I followed for Kitsune is slightly different in the way it treats with the core implementation of the problem (for example, in the way the behavior mechanics for the agents were implemented); and in this spirit, Kitsune could well be a first step for something much larger and efficient in the long term.

5.3 Future Work

There are a number of things that can be done to improve the quality and accuracy of the simulation in Kitsune, and as the time involved in Kitsune increased, ideas flourished.

The software was written in Java, a highly popular, cross-platform and flexible language, using the Swing library (widely used in mainstream applications). Furthermore, it has been placed under the MIT licence [14], which allows anyone to copy it, modify it and redistribute it, and the codebase is available to anyone on a SVN repository [15].Hence it would be very easy, for example for a future Brookes student, to build or improve on Kitsune.

The improvements that can be brought can be divided in 3 categories.

5.3.1 Improving the overall UI and the simulation workflow

The overall user interface, while functionnal, is far from ideal. It lacks visual feedback when tools are selected, and building editing operations could be made easier (for example by being able to draw lines of variable length instead of individual fixed-length segments, by allowing copy and paste in the editing zone, being able to undo and redo operations, import file from popular building designing applications, and so forth).

The simulation workflow itself too could use some refining. In the current state of things, the workflow is very straightforward (see figure 17). While it is sufficient for basic running simulations, it is very simple and improving it would enhance the flexibility of the software. The whole "simulation" part is especially static- the user cannot influence it at all.

Figure 17: Application workflow

Examples of additional possibilities in this spirit include:

- Editing of the agents' behavior during the simulation, and generally more flexibility in the agents' behavior
- Making the environment dynamic, for example being able to block some passages during the simulation (to simulate the collapsing of the structure during a fire)
- Being able to put out some portions of the fire during the simulation, to represent the action of firemen or fire sprinkler system.

An important part of the workflow is the gathering of metrics from the simulation. However, due to time limitations, this part of the application was not exploited as much as it could have. As of now, Kitsune presents very basic metrics. An obvious first step would be to increase the amount of data gathered during the simulation. For instance, in building evacuation, a crucial part that hinders the evacuation in some cases are bottlenecks. Bottlenecks occur when an area is occupied by too many people, and therefore their speed is limited, resulting in a slower evacuation. The identification of bottlenecks would be a very important feature for Kitsune.

Furthermore, the metrics are presented in a raw, textual form. User friendliness and readability could be improved by generating meaningful graphs from the collected metrics.

Finally, one feature that would be invaluable would be automated testing. To my knowledge, no building evacuation simulator offers this functionnality to this day. Such a functionnality would consist in the user designing of a building, and then having the program run automatically a series of tests, varying the placement and number of agents, fire sources, etc. After a defined series of tests have taken place, the program would use the collected metrics to automatically determine optimal placement for fire exits, doors, etc.

5.3.2 Improving the accuracy of the fire simulation

In the state of things, the fire simulation is rudimentary at best. While using a cellular automaton to simulate the propagation of a fire is a viable approach [16], the size of the automaton in our implementation is too small to reproduce accurately the behavior of the fire. An obvious method for solving this would be increasing the size of the fire grid. The algorithm could also certainly be refined, and a totally different approach (perhaps physics based) could be envisioned.

Another crucial aspect of the fire simulation is the smoke. In real life situations, smoke hinders visibility, and is the cause of many deaths in fire-related accidents. In our simulation, we did not take into account the smoke at all. While it can be modelized with a cellular automaton in a similar way to that of the fire, the propagation of smoke is a very complex subject (involving advanced physics equations such as Navier-Stokes'), and a lot of time would be required to implement a satisfactory smoke simulation.

5.3.3 Improving the AI of the agents

The whole point of this project was to simulate the evacuation of a building during a fire. Such a simulation relies heavily on the human reaction to the event. However, recreating this factor is very hard, if not impossible- however it is possible to have good approximations for it.

In the current state, only 3 basic behaviors are implemented: optimal behavior, follower, individual in panic. In order to increase the realism of the simulation, the number of available behaviors should be increased (we could consider for instance an individual that, despite the emergency, returns to a room to retrieve belongings or find another individual). This could be done either by implementing additional behaviors, or by integrating some sort of scripting support directly into Kitsune, using a language such as Lua. Additionally, we could envision dynamic behavior adjustment for each agent- under certain conditions, an agent would switch to another behavior (for example, a trained individual could panic if he doesn't find an exit after a certain amount of time).

Additionally, the agents do not take into account the presence of fire. They could adjust their itinerary depending on the intensity of the fire in certain areas of the building, amongst others.

Even though it is impossible to achieve a perfectly accurate reproduction of the human behavior, such adjustments would provide a much more precise sense of realism and detail into the simulation, resulting in more reliable results.

As we can see, a lot remains to be done with Kitsune. While it definitely does not consist in a solid alternative for professional evacuation simulation solutions, I believe that foundations have been layed, and that with the work of future undergraduate and graduate students from Brookes in the future, it could become a powerful and useful tool.

6 References

References

[1] U.S. Fire Administration. Residential structure and building fires. Website, 2008.
Published on http://www.usfa.dhs.gov/statistics/reports/residential-structure-fires.shtm.

[2] Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, 2002.

[3] Steeles. *Towards a Theory of Emergent Functionality*. MIT Press, 1990.

[4] NIST. Fifth international conference on pedestrian and evacuation dynamics. Website.
http://www.bfrl.nist.gov/info/PED2010/.

[5] MOLNÁR Péter VICSEK Tamás HELBING Dirk, FARKAS Illés. Simulation of pedestrian crowds in normal and evacuation situations. Website.
http://www.tu-dresden.de/vkiwv/vwista/publications/evacuation.pdf.

[6] HELIÖVAARA Simo EHTAMO Harri MATIKAINEN Katri KORHONEN Tim, HOSTIKKA Simo. Integration of an agent based evacuation simulation and the state-of-the-art fire simulation.
7th Asia-Oceania Symposium on Fire and Science Technology, 2007.

[7] HELIÖVAARA Simo EHTAMO Harri MATIKAINEN Katri KORHONEN Tim, HOSTIKKA Simo. Fds+evac: An agent based fire evacuation model.
4th Intl. Conference on Pedestrian and Evacuation Dynamics, 2008.

[8] TYAGI Gaurav. A heuristic optimization based methodology for fire evacuation simulation incorporating human behaviors. MSc. Thesis, 2004.
http://www.buffalo.edu/~zxue/Evacuation_Research/Research_papers/7-master-thesis-Tyagi.pdf.

[9] HUANG Rui DENG Zhihua YANG Lizhong, FANG Weifeng. Occupant evacuation model based on cellular automata in fire. *Chinese Science Bulletin Vol. 47 No. 17*, 2002.

[10] University of Greenwich. Fire safety engineering group. Website. `http://fseg.gre.ac.uk/exodus/air.html#build`.

[11] Savannah Simulations. Simwalk. Website. `http://www.simwalk.com/`.

[12] Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 1962.

[13] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *SIGGRAPH '87 Conference Proceedings*, 1987.

[14] OSI. Open source initiative - the mit license. Website. `http://www.opensource.org/licenses/mit-license.php`.

[15] Guillaume Ardaud. Kitsune google code page. Website. `http://code.google.com/p/kitsune-fest/`.

[16] J. Hearne S. Berjak. An improved cellular automaton model for simulating fire in a spatially heterogeneous savanna system. *Ecological Modelling*, 2001. `http://www-laep.ced.berkeley.edu/~itr/literature/paper/6cbuq8bi.pdf`.