CMP-7009A Advanced Programming Concepts and Techniques

Project Report - 13 December 2017

Emergency evacuation Simulator "Don't panic!"

Hateef Alshewaier, Nick Atkins, Zefeng Gao and Iana Gureva

School of Computing Sciences, University of East Anglia

Version 1.0

Abstract

The main goal of this project was to develop a piece of software which will simulate the emergency evacuation of a building in case of a fire. People within the building must correctly navigate to an evacuation assembly point from their current position. A Star algorithm was used for people's path-finding as the cheapest and fastest method for computing the shortest path. In order to visualise peoples movement the GLFW OpenGL library was used. The Win32 API was used to create buttons and dialog boxes.

The final application has a menu, a simulation window and a map editor. User can create their own map and run the simulation straight away. The map editor supports fire placement, multiple floor creation, building design and people positioning. At the end of the simulation special window appears displaying statistics (e.g. time in seconds).

Chapter 1

Introduction

1.1 MoSCoW

The main function of the application is to simulate the emergency situation. In case of the fire, everyone in the building must proceed to the exit. Thus, main feature of the program is moving individuals. The target point and pathfinding will help them to choose the right way. Fire spreading is essential part of the program as people should be affected by it during the simulation. Moreover, it is hard to follow constant movement without the controllable camera.

The evacuation area is always diverse, so the application should be able to read or create a map for each specific case. As the program is more science related, the used should be able to set up different scenarios and see how evacuation process is affected by the changes.

Although, simulation must look realistic and have the proper collision detection, high-poly 3D models of the surroundings will only slow down the calculations. Different high detailed features like fire extinguishers or constantly changing behavior are less important because simulation is a heavy computational process and without proper optimisation too many details can spoil the result.

Table 1.1: Features

Must	1. Assign each person an evacuation point which they must reach; 2. Crowd simulation; 3. Camera movement/zoom; 4. Use the A* algorithm for pathfinding; 5. Support collision detection between people and between people and objects (affects pathfinding); 6. Fire start point and spreading;
Should	1. Calculation for the unoptimized areas (stuck people); 2. Suggest improvements to buildings to speed up evacuation (e.g. more fire exits); 3. Include a configuration screen at the start for setting some variables (fire point, panic level, number of people, how fast fire spreads, speed of the simulation etc.); 4. Use a grid system with multiple floors; 5. Map reader; 6. Display statistics at the end of the simulation;
Could	1. Auto recognition of an object (room, door, fire exit, window, ladder, etc.) for map reader; 2. People make change their mind (change the group); 3. People could injure themselves and require help from others (affects their behavior); 4. Have a text note attached to people indicating status; 5. Include smoke effects; 6. Include fire extinguishers for people to fight the fire; 7. People dont know the exit 8. Do pathfinding for each person in parallel (multithreading);
Won't	1. Use texture mapping; 2. Include detailed models of objects (e.g. chairs, tables); 3. Support simulation of ¿ 100 people simultaneously;

1.2 Report structure

The first chapter in this report covers some background on human behaviour simulation methodologies and details some existing software to carry out these simulations. Chapter two explains the project management approach taken for the project and discusses the A-star path-finding algorithm used in the simulation. Implementation details specific to this application are explained in the third chapter including the modification of a third-party library and use of the singleton design pattern. Chapter four explains the testing methodology of the project and finally the last chapter summarises the results obtained and suggestions for future work.

Chapter 2

Background

2.1 Methodologies

There are many simulation methodologies that can be applied. Agent-based simulation (ABS) is one simulation that has done based on the behaviours in emergent situations. In fact, this simulation uses two aspects. First one is the Game of Life and the Purposes for this aspect is how the agents interact and what are the principles for that connections. Second, Boids models which focuses on the behaviour when the agents are collected and crowd[1]. Some simulations employ physical behaviour techniques to be combined into Virtual Reality (VR). The environment of Virtual Reality (VR) should offer the users tools which can be used for dive in imagination and be connected in 3D world [2].

Real time collision avoidance is one issue in many systems that apply behavior cases. The movement of each agent or element is always updating. All that because any agent must avoid collide with other agents. In fact, colliding is not an easy issue in simulation. Many ways are used in the case of how to control these agents. Tanner (2004) is one of actual approaches that covers behavior in action approaches of Balch and Arkin (1998), where is the statute of connecting between subsystems is located. That creates mass behavior between leaders and followers as in the approach of Tanner et al. (2004) [3].

2.2 Software review

Human population is always at risk from many hazards, but usually the biggest enemy to the human being is themselves. The demand in simulations that can play a tragic scenario without any actual people involved rose steadily during the past years. As the graphics capabilities became wider, several framework for the simulation were produced.

In November 2007, a multi-agent based framework for simulating human and social behaviour during emergency evacuation were developed. It has an editor function which allows user to assign predetermined positions to individuals and groups, or do it randomly. The framework is beneficial for group behaviour scientific studies as it models different types of human behaviour at microscopic level [4].

Another model is called BUMMPEE. An agent-based simulation that have individuals with disabilities support. It gives more accurate representation of the population, as the simulation can produce different results because of the dependency on individual criteria. Also, the model can classify the surroundings according to environmental characteristics. [5]

Chapter 3

Methodology

3.1 Project management

Each week our team had two meetings: Monday and Friday. During Monday hours, the pool of needed features was discussed. Most necessary tasks were selected and split between team members. After four days of implementation, another meeting point was established at the end of the week for work review and support if needed. That way if the problem occur the person could get an advice from others on early stages and proceed to do another work without carrying on unfinished assignments. Weekends were used to wrap up the work done and testing.

To help the team document iterations, Trello was used [6]. Every task had a card and a team member assigned to it. Three lists separated work in process, future tasks and the work done. All the documents had a free access to any team member. Comment section helped to keep in touch with current changes, so the opportunity to go back always existed in case of an emergency.

3.2 A star

In computer science, A star (A*) is an algorithm that computes optimal path between two nodes in a graph. The accuracy to performance ratio of this method make it the most suiting pathfinding algorithm for simulation. While the best known method, Dijkstra's algorithm, computes the shortest paths from one node to all other nodes in graph, A* have more narrow field of view what makes the calculations faster.

The order for nodes to be picked is resolve through heuristic function ($f(x)$). A* select the path, that minimises the equation (3.1):

$$f(x) = g(x) + h(x), \tag{3.1}$$

where x is the looked up node, $g(x)$ is the lowest cost for the path from the start point to x and $h(x)$ is a heuristic that estimates the cost of the cheapest path from x to the end point. Essentially, $f(x)$ is the length of the path, that is equal to the sum of the already traveled distance ($g(x)$) and the distance that is left ($h(x)$). That means, the smaller $f(x)$ is, the earlier we will get to the x point.

The time complexity of A* depends on the heuristic. Change of it can optimize the method, but the chosen $h(x)$ function must be suitable for the current situation. If the algorithm has only four ways to travel, Manhattan distance (3.2) will give the right result:

$$h(x) = |x.x - goal.x| + |x.y - goal.y|. \tag{3.2}$$

But if the diagonals exist as a valid option, Chebyshev distance (3.3) must be selected:

$$h(x) = max(|x.x - goal.x|, |x.y - goal.y|). \tag{3.3}$$

A* usage of the $f(x)$ function gives it an advantage over other path-finding algorithms because it goes through less nodes than all of them (in case of the same heuristic is used).

Chapter 4

Implementation

4.1 Graphical user interface and GLFW Modification

The application includes a graphical user interface to allow for easier interaction with the program (rather than typing on the command line). Buttons on the GUI scale depending on the size of the application window while drop-down and edit boxes are used to input data. The Windows Common Item Dialog is used for loading map files to help keep the appearance of the application consistent with the native Windows theme. Appropriate error messages are displayed if the user inputs invalid data or selects an invalid file to load.

The GUI is implemented as a singleton because exactly one instance of it is required while running the program. The instance is a static variable within the GUI class which has a private constructor and deleted copy and assignment operations. When the `initGUI()` method is called it first checks a bool variable which tracks whether or not it has been called before. If this is the first time initGUI() has been called then the instance is initialised by creating buttons and setting the "WM_COMMAND" handler function pointer for GLFW to GUIs `wmCmdHandler` function. `initGUI()` then returns a reference to the instance.

There were several possible approaches when implementing the graphical user interface. The first was to create GUI buttons as simple squares using OpenGL and check the position of the mouse on every click to determine whether or not it was located within the boundaries of the button. This would have allowed for basic GUI functionality without the need for additional third party libraries. The main disadvantage of this approach was that more complex GUI functionality such as drop-down boxes and input fields would have been much harder to implement. It would have also been difficult to make the button style match that of native Windows buttons.

The second possible approach was to use a C++ compatible widget toolkit such as wxWidgets or Qt. This would likely have resulted in the simplest implementation in terms of placing GUI elements in the window and assigning actions to them but would have resulted in the toolkit creating its own window separate from the GLFW window and OpenGL context. The GLFW documentation states that the window and context that it creates are inseparable and it is not possible to pass GLFW a handle to an existing window for it to link an OpenGL context to.

The final approach (and the one ultimately decided upon) was to modify GLFW to allow it to handle Windows API user interface elements. Windows uses a message-passing model to allow events from both the user and operating system to interact with a window. These events could be mouse button clicks, key presses or window closing to name just a few. The "windowProc" function inside the win32window.c source file of GLFW handles these messages using a switch statement to determine which action is taken depending on the type of message received (the WM_CLOSE message should close the window for example). All messages relating to placing GUI elements on the window and user interaction with them are "WM_COMMAND" messages however GLFW simply ignores these by omitting a case for them in the switch statement. The solution is therefore

relatively straightforward: add a case for "WM_COMMAND" messages in "windowProc" and pass GLFW a function pointer to a function in the applications GUI class which handles these messages. The following changes have been made to GLFWs source code:

1. A function pointer declaration - `_glfwWMCmdHandler(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)` has been added to the "win32_platform.h" header file.

2. A method `(glfwSetWMCmdHandler)` to set the above function pointer has been declared in "glfw3native.h" inside the "`#if defined(GLFW_EXPOSE_NATIVE_WIN32)`" block.

3. The above set method has been defined in "win32_window.c".

4. A case for "WM_COMMAND" was added inside the "windowProc" function of "win32_window.c". This simply checks that "`_glfwWMCmdHandler`" is not null and calls it passing the required parameters `(hWnd, uMsg, wParam, lParam)`.

4.2 A* optimisation

The original A* implementation used lists to track cells visited (closed) and those still to be processed (open). This meant that each list had to be searched for the next cell every iteration of the algorithm and caused the path-finding calculations to take up significant time (multiple seconds for 50x50 grids with multiple people). These lists were changed to maps of `pair<int,int>` to `GridCell*` allowing the next cell to be looked up in `O(log N)` time.

While the implemented A* algorithm worked well for a single grid, a typical building will have multiple floors that may have to be traversed in order for people to evacuate. A new algorithm was developed which, given the start and end grid cells as well as a list of stairs, allows a path to be built up recursively over multiple floors by first finding the following:

1. Paths between each set of stairs on each floor.

2. Paths from the start point to each set of stairs on the start floor.

3. Paths from the end point to each set of stairs on the end floor.

While it would be possible to do this all at run-time, the paths between stairs on each floor are pre-computed when a map is saved in order to reduce time spent calculating paths for each person when the application is running.

4.3 Running the program

4.3.1 Editor

The application includes an editor that allows the user to easily create and edit maps of buildings. Each building can have up to nine floors (only one exists upon creation of a building) which must all be the same size. The following objects can then be placed on the map by clicking their respective buttons on the left side of the window then on the cell that they wish the object to be placed. The available objects are:

1. Obstacles Represent walls, tables, plant pots or any other non-traversable object.

2. People The start point of a person when running the simulation.

3. Fire The start point of a fire when running the simulation.

4. Stairs Allows people to move from one floor (grid) to another, bidirectional.

5. End point The exit of the building which people must reach to evacuate successfully.

Stairs are placed slightly differently to other objects. The user must first click a cell on one floor and then a cell on a different floor to create a staircase between them. To remove an object from a cell simply click its respective button and then on a cell containing that object to remove it, no two objects may occupy the same cell.

4.3.2 Simulation mode

The simulation mode requires the user to load a previously created map file. They may then adjust the speed that people move and fire spreads at using the settings button, clicking on start will begin the simulation. A path is first calculated from each person to the exit point on the map. After this step the people begin to move and the fire begins to spread.

On earlier stages of product development program supported collision detection between people. However, only one person can be on cell at the time. This type of collision added a new issue to the human behaviour: the dead locks. If two or more people wanted to step on one cell at the same time, they got stuck. To solve this problem collision detection was switched to the collision avoidance algorithm. Cells within the grid obtained new variable: the queue. If cell was already taken by someone else, the person would wait for it to get free, hence step up to the waiting queue. The method helped to imitated basic human behaviour without hard computational powers.

Fire spreads around the map, choosing the neighbour cell that not an obstacle in random order and turning it into a fire. As in the real life, fire can spread to multiple floors. If the fire picks the staircase cell it will affect cell connected to it and than move randomly again to every possible direction.

Although, walls are not affected by it, the fire can hurt people. If a person moves through a cell containing fire during the simulation then their health will be reduced until it reaches zero and they die, people that reach the exit successfully are removed from the map. This type of the behaviour will model the situation more realistically as people cannot walk through fire without taking any damage. Health reserve will simulate different types of injuries going from a healthy person to dead one. The colour of the circle (which represents the human) will turn from green to yellow as they lose health. The red colour is an indicator that the person is hurt badly or, if he is not moving, not alive.

After the simulation is complete a results screen is displayed to show how many people survived and how many died as well as the total time taken for the evacuation. The timer value (in seconds) can indicate how fast and successful was the simulation. If the results are unacceptable, the user can reload the simulation and try to use different values or modify the map to model the perfect scenario. The pause button will temporarily halt the simulation and display the above statistics screen. That will help to keep a track on time and start again earlier if some results are insufficient. The timer will remain still during the pause and then will run again if the start button is pressed.

Before a file can be loaded (by either the simulation or editor) it must first be validated to prevent application crashes or other unwanted behaviour when provided with corrupt or incompatible data. Not only does the validator catch any exceptions that occur while checking the file but it also ensures that certain values (such as grid size) comply with restrictions within the program (minimum size 5x5, maximum size 50x50 in this case). If a file does fail validation then the application displays an error message to the user and remains on the main menu screen waiting for a new input.

Chapter 5

Testing

The programme was tested constantly during the whole implementation process. The protocol files consist of an identification number of the test case, steps describing executable process, expected and actual results. The "error" field helped to keep track of existing issues and divide them into two categories: major error (+), that breaks the program, or small problem (+/-), that does not affect the accuracy of the output . The "fixed" field has the mark if the problem was solved.

Some bugs were found on earlier stages (Appendix A: Test case 1, Figure 3; Appendix C: B1-B2) and some came up later (Appendix C: D10-D12). As the new features and windows were programmed, they were tested separately and then implemented into the application. Afterwards, the whole product was tested.

In the first version of the program user could pick the endpoint manually. The point position did not consider cell types, thus bugs like Appendix A: Test case 1, Figure 3 appeared. Subsequently, the map editor was created to stop user interrupting the simulation process and do all preparations beforehand.

The first version of the path finding algorithm had equivalent values for diagonal and straight movement. Although, cases like Appendix A, Test case 1, Figure 2 or Test case 3, Figure 2 is not a mistake in path computations, during the simulation the movement seemed unrealistic as people can not walk through walls. Moreover, the wrong values for movement caused the algorithm sometimes choose a non-optimal path (Appendix A: Test case 4, Figure 5). The method calculating the cost was changed so collision avoidance and path selection started work properly (Appendix C: A5, A10).

The interface testing discovered a few major bugs, that caused the program to crash if it was used inappropriately (Appendix C: D2-D3). All the fields containing user inputs were carefully restricted. The program ignores wrong values and displays the appropriate message for each field, explaining the borders (Appendix C: B2, B4).

Chapter 6

Results and Future Work

The results output by the program depend upon the layout of the map provided for the simulation but there are some general observations that can be made. The behaviour of people in groups is still not particularly realistic as people in the simulation tend to move in lines especially after pass through a narrow opening such as a doorway. More realistic behaviour would be to filter into the new room or corridor so that multiple people walk side-by-side if there is room to do so. More realistic behaviour is shown when approaching doorways because people in the simulation naturally bunch up while waiting their turn to pass through the narrow gap. Other improvements could be made to the path-finding such as attempting to avoid fire as it spreads throughout the building and by implementing a panic system for people that become trapped.

A lot of things from the section Could (Table 1.1) can be put as future work. The statistics window may show a little bit more information about the simulation. The coordinates of the problematic area could be detected and then displayed as a heat map on top of the simulation. The settings window could have a panic flag, that changes peoples behaviour (e.g. another heuristic function may compute more chaotic path for evacuation), the health of people might be adjustable as well as their speed.

The relations between individuals can be more mathematical and have influence on the behaviour of the whole group.

Bibliography

[1] Wai Kin Victor Chan, Young-Jun Son, and Charles M Macal. Agent-based simulation tutorial-simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. In *Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 135–150. IEEE, 2010.

[2] Fernando S Osório, SORAIA RAUPP Musse, Renata Vieira, Milton R Heinen, and Daniel C Paiva. Increasing reality in virtual reality applications through physical and behavioural simulation. In *Proceedings of the Virtual Concept Conference*, volume 1, pages 1–45, 2006.

[3] Silvia Mastellone, Dušan M Stipanović, Christopher R Graunke, Koji A Intlekofer, and Mark W Spong. Formation control and collision avoidance for multi-agent non-holonomic systems: Theory and experiments. *The International Journal of Robotics Research*, 27(1):107–126, 2008.

[4] Xiaoshan Pan, Charles S. Han, Ken Dauber, and Kincho H. Law. A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations. *AI & SOCIETY*, 22(2):113–132, Nov 2007.

[5] Keith Christensen and Yuya Sasaki. Agent-based emergency evacuation simulation with individuals with disabilities in the population. *Journal of Artificial Societies and Social Simulation*, 11(3):9, 2008.

[6] Zefeng Gao Iana Gureva Hateef Alshewaier, Nick Atkins. Project management: Trello. `https://trello.com/b/sdojQFti/emergency-evacuation/`, 2017. [Online;].

Contributions

Table 6.1: Contribution

Hateef Alshewaier	Initial (unused) group beahviour and colision detection, background methodology section of report, delete people after reaching exit, some testing. 5%
Nick Atkins	Report sections: Report structure, Graphical user interface and GLFW modification, A* optimisation, Running the program (Editor and part of Simulation mode), first results paragraph. Code: Translate A* implementation from Java to C++, majority of OpenGL code, GUI except settings button/dialog box and create map dialog box, file picker, multi-floor path finding, bug fixing, GLFW modification, file validation. 45%
Zefeng Gao	Map creation dialog box and menu screen. 5%
Iana Gureva	A* in Java, people movement and health, fire spreading, file loader, settings window, statistics window, testing, majority of report (all sections not done by Nick Atkins). 45%

Appendix A

Appendix B

Appendix C