Simulating Pedestrian and Traffic Interaction for Metropolis

Arthur Vivian
B.A.(Mod.) Computer Science
Final Year Project April 2011
Supervisor: Dr. Carol O’Sullivan

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

____________________________________________________                 ___________________
Name                                            Date
Acknowledgements

I would like to thank my supervisor, Carol O'Sullivan and those working in the GV2 research group for all the help provided in doing this project.

I would also like to thank my friends and family for all the support given to me over the past year during this project. I'd especially like to thank my fellow classmates for all their continuous help and kind words.

Finally, I would like to give a very special thanks to my tutor, Lucy Hederman, for help me make (hopefully) the right choice this year!
Abstract

Simulating the real world is a continuous area of research both in within and outside of Computer Science. There currently exists such a simulation being built by the GV2 research group in Trinity College Dublin, which aims to recreate a living, breathing city in which a real person can move around a realistic virtual world. There currently is little interaction in this world between the pedestrians and vehicle, both of which it simulates. This project investigates the how to best simulate Pedestrian-Vehicle interaction in such a virtual world, with particular focus on queues at bus stop.
## Contents

<table>
<thead>
<tr>
<th>Chapter 1 – Introduction</th>
<th>p.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Motivation</td>
<td>p.6</td>
</tr>
<tr>
<td>1.2 Objectives</td>
<td>p.6</td>
</tr>
<tr>
<td>1.3 Overview of Report</td>
<td>p.7</td>
</tr>
<tr>
<td>Chapter 2 – Background</td>
<td>p.8</td>
</tr>
<tr>
<td>2.1 Metropolis</td>
<td>p.8</td>
</tr>
<tr>
<td>2.2 Current Pedestrian-Vehicle Interaction in Metropolis</td>
<td>p.9</td>
</tr>
<tr>
<td>2.3 Current Research into Pedestrian-Vehicle Interaction</td>
<td>p.10</td>
</tr>
<tr>
<td>2.4 Other Examples of Simulated Pedestrian-Vehicle Interaction</td>
<td>p.11</td>
</tr>
<tr>
<td>2.5 Finite-State Machines</td>
<td>p.11</td>
</tr>
<tr>
<td>Chapter 3 – Design</td>
<td>p.13</td>
</tr>
<tr>
<td>3.1 Considerations</td>
<td>p.13</td>
</tr>
<tr>
<td>3.2 Observations on Pedestrian Queueing Behaviours</td>
<td>p.14</td>
</tr>
<tr>
<td>Chapter 4 – Implementation</td>
<td>p.16</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>p.16</td>
</tr>
<tr>
<td>4.2 Locations</td>
<td>p.17</td>
</tr>
<tr>
<td>4.3 Agents</td>
<td>p.19</td>
</tr>
<tr>
<td>4.4 Agent State Machine</td>
<td>p.22</td>
</tr>
<tr>
<td>4.5 Message passing between Locations and Agents</td>
<td>p.24</td>
</tr>
<tr>
<td>4.6 Collision Detection between Pedestrians in Queues</td>
<td>p.25</td>
</tr>
<tr>
<td>Chapter 5 – Analysis</td>
<td>p.26</td>
</tr>
<tr>
<td>5.1 Testing in own Virtual Environment</td>
<td>p.26</td>
</tr>
<tr>
<td>5.2 Evaluation</td>
<td>p.30</td>
</tr>
<tr>
<td>Chapter 6 – Conclusion</td>
<td>p.32</td>
</tr>
<tr>
<td>6.1 Future Work</td>
<td>p.32</td>
</tr>
<tr>
<td>Bibliography</td>
<td>p.33</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

From use in urban planning and assistive technologies (such as planning for pedestrian queues and building system to help the blind at traffic lights) to uses in computer games (for example, the Grand Theft Auto series), the simulation of virtual cities has many applications and is of interest to those both outside of and within the Computer Science domain. For the user to experience complete immersion into this world, accurate simulation of the environment is needed. A major part of this simulation includes the interaction between Pedestrians and Vehicles, as they are two prominent elements of a city, as well as being of the most interest to those who wish to simulate such a city. The Metropolis project currently being undertaken by the GV2 research group is hoped to create such a simulation of a city and the research done in building Metropolis is hoped to be of use to those wishing to build other virtual cities in the future. Such a project involves an immense amount of time and work in order to build an accurate representation. This project hopes to contribute to Metropolis by looking at how interaction between Pedestrians and Vehicles can be performed and then implementing this into Metropolis. Metropolis currently has vehicles and pedestrians in the application, however there is little to no interaction between them. It is hoped that this project can go some way to changing this.

1.2 Objectives of Project

The main objectives of the Project are:

1. To build an accurate simulation of an interaction between Vehicles and Pedestrians that can be used in Metropolis or any other virtual world simulation.
2. To build this simulation in such a way that others in the future can easily modify this simulation to either improve upon it or to easily add in additional features.
3. To gain an insight into the world of graphical A.I development and hopefully take something worthwhile away from it.
1.3 Overview of Report

The report has been divided into the following six chapter:

1. **Introduction** – This chapter discusses the motivation behind the project, the objectives for the project and give a layout of the report.

2. **Background** – An Introduction to Metropolis and its current state is given, which attention given to what has been achieved in relation to Pedestrian-Vehicle Interaction. Previous work on simulation pedestrian and vehicle interactions are also outlined.

3. **Design** – This Chapter looks at some of the main factors which helped shape the design of the project.

4. **Implementation** – This chapter will go into some detail into the main components of the project and how they were built to fit in with one and other.

5. **Analysis** – A look at how the project was tested and evaluated, including feedback from a group who were shown the final result.

6. **Conclusion** – This chapter will look at what was achieved in this project and look at possible future work that could be implemented.
Chapter 2

Background

The Metropolis project has been in development for some time. This chapter will discuss the current state of the Metropolis system. It will also look at some of the current work that has been done in simulating pedestrian interactions with vehicles.

2.1 Metropolis

Metropolis is a research project currently being developed by the GV2 research group in Trinity College Dublin. It is a novel interdisciplinary project combining computer graphics, engineering and cognitive neuroscience research where the aim is to create a simulated real-life city, where real people will be able to move around and experience a computer generated Virtual Dublin.[website]

The project started in 1997 with funding from Science Foundation Ireland. As well as GV2, the project is also being built in collaboration with the TCD Institute for Neuroscience and the TCD Department of Mechanical and Manufacturing Engineering. It is envisaged that the project will be of practical benefit to urban planning projects, the development of assistive technology for people
with disabilities as well as computer games.

Five of the main research objectives of Metropolis that relate to this project are:

- Apply principles of human multi-sensory perception to create the most realistic large-scale simulations of populace ever realised.
- Increase the realism of these large crowds of people by adding variety in animation, appearance and sound, driven by perceptual models and metrics.
- Add real meaning to the simulations by endowing individual crowd members with appropriate, sentient behaviours that are based on cognitive and sociological models.
- Simulate realistic populace and traffic noise, effectively propagated depending on environmental factors, and driven by psychoacoustic principles.
- Provide easily integrated populace and traffic for Virtual Environments that will be scalable: at the architectural level (i.e., from PC to large cluster exploiting multi-core architectures); the user level (i.e., from single user to massively multi-player); and the crowd level, (i.e., millions of people simulated in real-time). [1]

It is hope that this project being under take can help metropolis further achieve those 5 aims.

2.2. Current Pedestrian-Vehicle Interaction in Metropolis

Metropolis currently simulates pedestrians moving through a Virtual Dublin. Pedestrians walk throughout the city, sometimes congregating together in have conversations. Motion capture animation has been added to the models to make them look as realistic as possible. This includes conversational gestures, as well as walking motions of various speeds. Audio is also used in the conversations. Pedestrians move around the city via a series of walk maps which have been developed by hand by researchers in GV2.

Traffic simulation includes vehicles moving along roads using a system of nodes. To navigate, vehicles look for the next node along a path of nodes. Nodes can be set to be a stop point, so when a vehicle reaches such a node it will wait at the node until told it can move on. Metropolis provides a
graphical editor to help edit street maps. Below is a picture demonstrating the graphical editor.

Metropolis also features physical properties for each of the simulated vehicles. This includes collisions between vehicles. Vehicle AI is run by a fuzzy logic controller, written by GV2 researcher, Anton Gerdelan. The vehicle models range from simple cars to accurate representations of Dublin Buses, with each model having its own attributes, such as the physical size of the vehicle body, the size and positioning of the wheels, the torque of the engine and gear information.

There is no actual interaction between pedestrians and vehicles in Metropolis though. While vehicles do have the ability to stop at traffic lights and pedestrians will walk over zebra crossings, vehicles will simply drive straight through a pedestrian, as if it were a ghost. Pedestrians also have no concept of pedestrian crossings, and will not stop and wait when the crossing is in a state for pedestrians to do so.

2.3. Current Research into Pedestrian-Vehicle Interaction

A number of papers have been written to suggest models of pedestrian-vehicle interactions. As discussed later on in the report, the scope of this project was narrowed down to focus on the queuing of pedestrians at bus stops. A paper on this subject was written by Okazaki and Matsushita, entitled *A Study of Simulation Model for Pedestrian Movement with Evacuation and Queueing*. It concentrates on three main types of pedestrian queues, queues in front of counters, queues in front of gates, and queues in front of elevators, which is of most interest to this project, as they are similar if not identical to queues in front of bus doors. The model they suggest is based around magnetic forces acting on pedestrians. Each pedestrian has a positive forces associated with it, with its destination having a negative force. Wall and other such obstacles, as well as pedestrians, have positive forces. Pedestrians will be pulled towards the negative force and be repelled away from positive forces. The papers simulates queues in front of elevators in this fashion a proves that it is
2.4. Other examples of Simulated Pedestrian-Vehicle Interaction

There are many examples of simulated virtual worlds built to display Pedestrian-Vehicle interactions. These range from software suites built to aid the design of urban spaces to computer games. These programs were looked at for inspiration in the design of this project. Some of the prominent examples are discussed below.

Quadstone Paramics are a company that specialize in microsimulation modelling software. Their products include functionality to simulate pedestrian crossings and public transport terminals. They claim to have over 1000 companies in over 45 countries that use their software. [3] This shows that there is a definite need for such simulations out there.

Legion are another company that have a range of software dedicated to microsimulation. Examples of where software has been used in the construction of the Olympic Park in Sydney, Australia and the 34th Street Subway Station in New York. [4]

Another area with many examples of simulated pedestrian-vehicle interaction is that of Computer Games. The grand theft auto series has perhaps, though quite controversial, been one of the pioneers of simulating pedestrian-vehicle interaction. As early as 1995, DMA, the company behind Grand Theft Auto, went about figuring out how to create such interaction in a realistic manner, for a game called “Race'n'Chase”, which later became Grand Theft Auto 1. [5]

2.5 Finite-State Machines

As such a simulation relies heavily on Artificial Intelligence, a brief overview of Finite State Machines, which are used in the project in order to control the decision making of the Agent in the simulation, will now be given.
Finite-state machines, or finite-state automata, which shall be referred to though out this report as simply state machines, are a mathematical abstract used in the design of digital logic and computer programs. One of their main applications is in the use of Artificial Intelligence. State Machines are a behaviour model composed of a finite number of states, transitions between those states and actions which are performed on leaving or entering a state. As well as having a finite number of states, they have a method for input, and a method for output. The operation of a finite state machine begins at one of its states, referred to as a start state and goes through different transitions to other states depending on input at its current state.

The current state of the state machine is determined by its past states. As such, it can be said that the current state records information about the past, i.e., it reflects the input changes from the system start to the present moment.

A transition occurs when there is a state change and is essentially the condition that would need to occur to enable the transition.

An action is a description of an activity that is performed at a given moment. These actions could include entry actions, which are performed when a state is entered, exit actions, which are performed when a state is entered and transition actions, which are performed during the transition between states.

The following is a diagram of a simple finite state machine used to control a light switch. The format of the diagram will be used for other finite state machines throughout the report.
Chapter 3

Design

This chapter will look at some of the factors which helped shape the design of the project.

3.1. Considerations

Before undertaking the project, a few design choices had to be considered. First of all was the scope of the project. There are potentially quite a lot of possible interactions that can occur between vehicles and pedestrians. Some of these identified include:

- Pedestrians walking across zebra crossings, vehicles waiting.
- Pedestrians jaywalking across the street, vehicles stopping abruptly.
- Pedestrians waiting for or hailing down taxis.
- Accidents between Pedestrians and Vehicles (collisions).
- Pedestrians queues at transport stations (Bus, Tram, Train, etc).
- Vehicles parking, drivers leaving vehicle and becoming pedestrians.
- Pedestrians mounting/demounting bicycles.

As well as all these interactions are the appropriate behaviours to be considered for each. For example, when pedestrians are waiting at zebra crossings they may form a queue in a specific manner, namely lining up along the street. Other pedestrians may either join the queue by walking up to a free area at street level or by waiting behind other pedestrians. Some pedestrians may also be impatient and try to get to the front of the queue.

It was decided that the project should concentrate on the queue of pedestrians at transport stations. Focus was put on bus stops, as they have similar properties to train station platforms and tram stops, and so any work done in investigating how to simulate bus stop queues could be easily utilised again. Since Metropolis already featured buses and had some bus stops being rendered as well, these seemed like a good place to start.

Metropolis has been under development for some time, since 2007. The code base at the point of beginning this project had become quite large and complex. It was decided that rather than work
directly on the Metropolis code base, the package would be created containing all the A.I and functionality. This would be coded in C++ (as Metropolis was also coded in C++) and could then be inserted into Metropolis at a later date. An advantage of this is that the package could also be inserted into any other Virtual World simulation. This meant that the overall design of the package had to be carefully considered, so it would be flexible enough to be able to mould itself around another piece of architecture, but ultimately be able to work with Metropolis.

3.2. Observations on Pedestrian Queuing Behaviour

In order to build a model for simulating pedestrian queues, it was first necessary to investigate how exactly pedestrian queues are formed and what behaviour pedestrians they exhibit once in them. This investigation was done by sitting and observing behaviours at bus stops. The main bus stop that these behaviours were observed was a Nassau Street outside of Trinity College Dublin. This stop was specifically observed as it was already included in the Metropolis. Filming a sample of bus stops was considered for this part of the project, however access to such equipment would have been difficult and there may have been ethical issues involved in filming such scenes.

Many key behaviours where observed that helped model the pedestrians queues. These include:

- The Idling behaviours of pedestrians when the are waiting for a bus to arrive at a stop. Such behaviours include playing with mobile phones, reading newspapers/magazines and pacing up and down while waiting for the bus.
- The positioning of pedestrians as they wait for a bus to arrive. This positioning is usually somewhere where they can stand or rest easily. Often a wall is used to lean against, either to rest or to stay as far back off the pavement in order to let other pedestrians pass. Occasionally they will wait by the stop sign.
- Buses usually stop at the Bus Stop sign, and queues form at the location of the bus stop door. So once a bus has come into view of the bus stop, pedestrians will begin to move towards a bus stop sign. However, queues are not usually formed until the bus has fully stopped, as pedestrians can be unsure as to where the bus might actually stop.
- Because of the uncertainty of where to begin forming a queue, actually queues tend to be quite chaotic in nature. Lines can be formed, however there is more of a sense that who ever is closer to the door is the next one to be let on by the other members of the queue. This doesn't always happen however, occasional someone will push past to get the front. These
occurrences are rare.

- Those queuing will generally let those on the bus get off first. This can mean the crowds in front of a bus door will now have to move away to allow those on board to come off, or that crowds do not form in front of bus doors at all.

- Pedestrians can join the queue at any time. For example a pedestrian can be late to get to the bus stop and so will spend no time waiting at the stop idling, rather walk, or run, straight to the queue.

These behaviours where taken into account when designing the model for pedestrians queues.
Chapter 4
Implementation

This chapter will discuss how the project was built and look at the overall architecture of the package built for Metropolis. It will start by giving a quick overview of how the system works before going into a more in depth look at the individual components. Code samples are given to help demonstrate the architecture of the code. Please not that while these code samples contain the same functionality as that in the package, they may differ slightly and some extraneous code has been omitted. Please also be aware that there may be some “Chicken before the Egg” in describing the architecture, though effort has been put in to avoid this.

4.1. Overview

There are two main concepts within the system. These are Agents and Locations. An Agent is a state machine and represents any independently moving entity on the screen. This includes Pedestrians and Vehicles. Locations are areas in the virtual world which agents move to. There are also a means by which two different types of agent can communicate. Essentially, all the interaction between Pedestrians and Vehicles is done through Locations. Based on the state of the agents, the locations will communicate different messages to the other agents when interaction is needed. So for example, when a PedestrianAgent reaches a BusStopLocation, the PedestrianAgent will let the BusStopLocation know it has reached it and will then change its state from one of walking to one of idling. Pedestrians form queues by moving towards a TransportLocation which is normal a representation of a vehicle door. If they collide with each other, their appropriate A.I, depending on the state will decide how to effectively for a queue.
4.2. Locations

Within the package there is a base class `Location` from which more specific types of `Location` can be inherited from. Below is the class declaration of `Location`.

```cpp
class Location{
private:
    Point2 pos;
    locationType type;
public:
    Location(){
    Location(void *);
    Location(Point2 _pos, float _radius, locationType(_type))
        : pos(_pos), type(_type){}
    Point2 getPos();
    void setPos(Point2 newPos);
    locationType getLocationType();
    virtual bool pedAtLocation(PedestrianAgent * ped){
    }
};
```

We see that the `Location` class contains information on the position of the Location in the world as well as they type of Location. It also contains a virtual function call `pedAtLocation`. This function is described by inherited class to let a pedestrian know when it has reached that location. Different locations will have different reasons defining whether a pedestrian has reached it. For example, a Bus Stop Location type may just be a square and a pedestrian will have reached the location once inside the square.

Two classes which inherit from `Location` that are vital to the package are class `WaitLocation` and class `TransportLocation`. The class definition for `TransportLocation` is given below.

```cpp
class TransportLocation : public Location{
private:
    std::vector<PedestrianAgent*> peds;
    int pedsOnBoard;
public:
    TransportLocation(){
    ~TransportLocation(){
    TransportLocation(Point2 _pos)
        : Location(_pos, 0.3f, TRANSPORT), pedsOnBoard(0){
    void addPedToTransport(PedestrianAgent * ped);
    void updatePos(Point2 nPos);
    bool pedAtLocation(PedestrianAgent * ped);
    void pedGetOff();
    void updatePedsOnBoard(int nPeds);
    int getPedsOnBoard();
};
```
TransportLocation represents the location of a vehicle which a pedestrian is interacting with. It may be that the pedestrian is simply moving towards this vehicle or, in the case of a bus for example, the pedestrian maybe on currently on the bus and so will need to be hidden from the screen. We can see from the class definition that the class contains a vector which contains pointers to PedestrianAgents. This is allow the Location to communicate with the pedestrian currently at the location.

The class definition for WaitLocation is follows.

```cpp
class WaitLocation : public Location {

private:
    std::vector<PedestrianAgent*> pedsWaiting;
    Box pedWaitArea;

public:
    WaitLocation(){};
    WaitLocation(void*);
    WaitLocation(Point2 queuePos, Box _stopArea, Box _pedWaitArea);
    ~BusStopLocation(){}

    bool pedAtLocation( PedestrianAgent * ped);
    Point2 getVehiclePos();
    Vector2 getPedLocation();
    int getPedAmount();
    Box getStopArea();
    void addPedToQueue( PedestrianAgent * ped );

};
```

WaitLocation represents an area in the world at which a Pedestrian agent can wait. Such locations may include a Bus Stop or a Pedestrian Crossing. WaitLocation also contains a vector which holds pointers to Pedestrian Agents.

From these classes, other classes can be derived that more accurately represent a certain type of Location. An example of such a class hierarchy is given below.
We see here that a theoretical PlatformLocation class has been derived to represent a platform at a train station. This class could have information to describe a platform location that a normal WaitLocation class wouldn’t contain.

### 4.3. Agents

A Class called Agent represents all of the independently moving entities on the screen. These Agents move based on their own individual A.I. The A.I. in the Agent is controlled though state machines. These state machines will be discussed later on in the chapter. This class is a base class from which more specific types of agent can be derived. A class definition of Agent is given below.

```cpp
class Agent
{
    private:
        Point2 pos;
        Vector2 dir;
        bool dead;
        agentType type;

    public:
        Agent()
        Agent(Point2 _pos, Vector2 _dir, agentType _type);
        ~Agent()
        void updatePos(Point2 newPos);
        virtual int collide(Entity * other) {}
        virtual void update(int timeElapsed) {}
        virtual bool isDead(){return dead;}
};
```

From the class definition, it can be seen that there is a function to detect a collision between other entities in the world. These collisions will be discussed in more detail later on in the chapter.
One of the main types of agent that the package depends on is PedestrianAgent. A class definition of PedestrianAgent is given below.

```cpp
class PedestrianAgent : public Agent {
private:
    Location * location;
    bool atLocation;
    bool atDest;
    bool visible;
    Point2 dest;
    pedState state;
public:
PedestrianAgent();
PedestrianAgent(void*);
PedestrianAgent(Point2 _pos, Vector2 _dir) : Entity (_pos, _dir, PED);
~PedestrianAgent(){}
void update(int timeElapsed);
void draw();
int collide(Agent * other);
bool isDead();
pedState getState(){ return state; }
Point2 getDest(){ return dest; }
Vector2 getLocationPos();
bool isVisible(){ return visible; }
bool isAtLocation();
bool isAtDest();
void setVisible(bool v);
void setDest(Point2 newDest);
void setState(pedState newState);
void updateLocation(Location * newDest);
};
```

Pedestrian Agents move throughout the virtual world by moving towards locations. Once a Pedestrian agent has reached a location, it will update that location to let it know it has arrived and the Location, depending on the type of location, will react in an appropriate way. More specific types of these interactions will be discussed later in the chapter. Pedestrian agents interact with other Pedestrian agents, for example to avoid collisions with each other. However they do not interact directly with other forms of agent.

The VehicleAgent class represents a base class for different types of Vehicle. More specific types of Vehicle can be inherited from this class, for example a bus or a train. A definition of this class is given below.

```cpp
class VehicleAgent : public Agent{
private:

```
TransportLocation doorLoc;
vehicleState state;
public:
VehicleAgent();
VehicleAgent(void*);
VehicleAgent(Point2 _pos, Vector2 _dir)
: Entity (_pos, _dir, VEHICLE), state(inMotion);
~VehicleAgent(){}
void update(int timeElapsed);
void draw();
int collide(Agent * other);
bool isDead();

Since the project concentrated on the queuing of pedestrians at bus stops, it seems appropriate to
discuss a BusAgent Class that was create for the purposes of the project. The class definition for
BusAgent is below.

class BusAgent : public VehicleAgent{
private:
    Point2 doorleftpos;
    Point2 doorrightpos;
    BusStopLocation * busstop;
    busState state;
    int counter;
    int secs;
public:
    BusAgent();
    BusAgent(void*);
    BusAgent(Point2 _pos, Vector2 _dir,
        BusStopLocation * _busStop)
        : Entity (_pos, _dir, BUS), speed(BUS_speed),
        busstop(_busStop), state(inMotion);
    ~BusAgent(){}
    void update(int timeElapsed);
    int collide(Agent * other);
};

A Bus Agent, similar to PedestrianAgents, move through the world by travelling towards locations.
However they look specifically for BusStopLocations to move towards.

Below is a sample class hierarchy diagram that could be used in a simulation.
As well as a BusAgent Class which was shown earlier on, other classes can easily be derived from VehicleAgent.
4.4 Agent State Machines

Given below are diagrams describing the state machines of the bus and pedestrian agents.

**PedestrianAgent State Machine**

Pedestrians are in a walking state as they move around the world. Once they have reached a Location they then enter into a Idle state. Within this Idle state, idling behaviour can be added to the system, such as a pedestrian using a mobile phone, or a pedestrian pacing between different spots waiting for a bus to arrive. Once a bus arrives at the stop, the bus stop will signal the to the pedestrians to begin moving towards the entrance to the bus. The Pedestrians will at this point enter into a queueing state. Once they have reached the door of a bus, they will enter that state of OnTransport, where they will not be visible on screen and collisions between that pedestrians and others will be ignored. Pedestrians will enter back into a walking state once they have left the transport.
The BusAgent state machine is quite a large and complex machine compared to that of the pedestrian. As a bus moves along a road it remains in a state inMotion. Once it reaches a certain distance to the bus stop, it will alert the bus stop that it is now near and change its own state to nearStop. Once a bus arrives at the bus stop itself, it will go into a state where its doors will begin to
open. Once opened, the bus will begin to let pedestrians off the bus. Once all these pedestrians have left the bus, the bus will change state to allow pedestrians to board. It will alert the bus stop to this who will inform the pedestrians to start moving towards the location of the bus door. Once all the pedestrians have boarded the bus, the bus will change its state to close its doors and the bus will begin to move away from the stop.

### 4.5 Message passing between Locations and Agents.

Interaction between pedestrians and vehicles takes place by passing messages to each other between locations. This message passing is done by alerting locations to changes in their states, which will alert the other agent to this change in state. An example of this interaction between a pedestrian agent and a bus agent is given in the diagram below. Please note this shows a simplified version of the state machines.

![State transition diagram](image)

A short explanation of the diagram follows:

- As the bus moves from an inMotion state to the of nearStop, it sends a message to the bus stop telling it it has changed states, and the bus stop will in turn let all the pedestrians currently waiting at the stop know that they are to begin the queueing process.

- Once the bus reaches the stop and is fully stopped and ready to let pedestrians on board, it will again alert the bus stop of its state change who will alert all the pedestrians queueing that they may now begin boarding the bus.

- Once all the pedestrians have boarded the bus and left the queue, the bus stop will alert the
bus to let it know there are no more pedestrians waiting and the bus will begin to move away from the stop.

4.6 Collision Detection between Pedestrians in Queues.

Initially, the queueing of pedestrians was performed by creating a QueueList data structure which would work like a linked list to hold the pedestrians in their positions in a queue. Once a node ahead of a pedestrian in the linked list became free, it would be able to simply move to the position of that node. This, however, produce results that appeared too “utopian” in nature. It was decided that in order to create realistic looking queue behaviour, each pedestrian should essentially have a “mind of its own” and be able to move about the world freely in search of its next location, similar to the ideas discussed in Okazaki’s and Matsushita's paper. When pedestrians queue for a bus, they do so by all trying to move towards the location of the bus door. If they are the only pedestrian within the area, they should be able to simply walk up and on to a bus. However, this is rarely the case, and so a form of A.I. was built for when pedestrians collide with each other in the virtual world. A collision occurs when two pedestrians are within a certain distance of each other. However this can be easily changed to allow for more complex collision detection methods. A simplified algorithm showing what happens when two pedestrians, both in queue states collide is given below:

- If there is a collision between two pedestrians, A and B, and they are both in the queue state.
  - If pedestrian A is closer to the bus stop than pedestrian B,
    - then pedestrian B waits for pedestrian A to move out of its position and towards the stop before advancing towards the stop.
    - Other wise pedestrian A waits.

This simple form of A.I. has shown to be effective in creating realistic behaviour, as will be discussed in the next chapter.
Chapter 5

Analysis

This Chapter will discuss how the package containing the Pedestrian and Vehicle A.I. was tested and evaluated and a discussion on the results will follow.

5.1. Testing in own Virtual Environment

The package that was created was used within a virtual environment that was coded using OpenGL. This environment simulated pedestrians moving about on the street and moving towards a bus stop. Once they reached the bus stop they would being to wait at a certain area.

Many of the considerations talked about in Chapter 3 where implemented into this Virtual Environment. The average walking speed of a human ranges from 4.51km/h to 5.43km/h [6] and so it was decided that an average speed of 5km/h would be used for the simulation. The dimensions of a Wright Eclipse Gemini bus chassis where chosen as the dimensions of a Dublin bus, as these chassis’s are widely used in many of the newer Dublin bus fleet. These measurements are shown below.

The environment was created with a single straight road. On either side of this road was a footpath. Roughly halfway down one side of the road a bus stop was located. On either end of the footpath spawn points for pedestrians were placed. Pedestrians spawn at these points at random intervals and either make their way down the street or towards the waiting area of the bus stop.
The bus stop was designed with a wall at the back of the footpath and measured roughly to represent the bus stop on Nassau Street outside the entrance of the Trinity College arts block. This was done so that when evaluation of the package needed to be done, it would give those who were looking at it something familiar to relate it to.

The simulation shows a bus spawning at one end of the road. The bus then makes its way towards the bus stop where it stops and waits for pedestrians to board. Once all the pedestrians have boarded the bus then moves away. Once it has reached the end of the road it then respawn at the beginning of the road. The bus keeps all of the pedestrians that boarded the bus. The bus will once again travel towards the bus stop where it will again wait and let off all of the pedestrians and then let on the other pedestrians currently waiting at the stop.

The above picture shows a screenshot of the road running through the world of the program. Note the pedestrians walking down the pavement and the yellow bus stop sign,
The above image shows a representation of a pedestrian used in the simulation.

The above image shows the model of the bus used in the simulation.
The above image shows pedestrians queuing as they are about to board a bus.

5.2 Evaluation

Evaluation on the project was done by showing the simulation of queuing pedestrians to a group of adults who had all used Dublin Bus before. They were each given a small form to fill out to find out their age and background and then shown a short clip of the simulation taking place. They were then asked to rate from 1-5 how realistic they thought the simulation looked compared to an actual real life bus stop. They were next asked whether the bus stop looked like a bus stop they were familiar with in Dublin. Finally they were asked to comment on why they gave the rating they did.

A total of 11 participants were shown the demonstration. The average rating for the clip came to 3.4 out of 5. This shows that while not perfect, the demonstration didn't perform too negatively. Feed back on this demonstration itself was quite mixed. Some of the participants thought that the queueing behaviour looked quite realistic, and even stated that the stop shown looked that of
Nassau Street, where most of the behaviour for the simulation was observed. Others thought that it was too chaotic. One girl from Sweden informed me that in Sweden queues at bus stops are far more orderly, with pedestrians queuing in a line, and with new pedestrian entering this queue at the end in a neat fashion. This opinion was shared with several of the other participants, who states that the simulation, while does look realistic in some cases of bus stops, may be overall to chaotic to be a some what represent all queues at bus stops. Another participant from Tallaght told me that the simulation was not chaotic enough, as people from Tallaght generally push past each other to get on to a bus.

From these results it can be clearly seen that pedestrian queues can vary quite dramatically. There seems to be many factors underlining that result in a realistic representation of a queue, many of these seem to be cultural. It could then be suggested that more effort should have been put into researching these cultural aspects and how they could have been worked into the model.
Chapter 6

Conclusion

This chapter will look at possible future work that could be implemented into the project. Learning outcomes of the project will also be discussed.

The project has resulted in a model in which Pedestrian-Vehicle interaction can be simulated. Classes have been built to represent Locations and Agents and these can easily be inherited from to form new variations of locations and agents. Lessons have been learnt in how to successfully simulate believable pedestrian queues at Bus Stops, which the cultural aspects of a queue needing to be fully considered before setting out. It may be suggested from this project that other factors in human nature could also be considered in how people queue, such as how courteous one pedestrian may act towards another pedestrian, whether that be based around the age of a pedestrian or just depending on how bad a day that pedestrian is having.

6.1 Future Work

There are many possibilities for future work in this project, as there are many forms of pedestrian-vehicle interaction. Currently the package has the functionality to simulate pedestrians entering a leaving buses at bus stops. This work could be modified to allow for other similar pedestrian-vehicle interactions, such as the queueing of passengers about to get onto a train or at a tram stop. All that needs to be factored in to allow for these sorts of interactions is the fact that there are multiple doors on trains as a posed to just a single door on a bus.

The inclusion of Zebra Crossings could also be implemented. There is currently a crossing location class defined, with a state machine defining the various states a zebra crossing goes through already in place. There is just a small amount of work still required in communicating information between the crossing and Pedestrian and Vehicle Agents needed to be done in order to have zebra crossing fully functional.

Since the aim of the project was to implement what was produced into Metropolis, it would be nice to try and see if the package produced would be able to slot in effectively. This was not carried out during the course of the project due to time constraints.
Bibliography


