ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΣΧΕΔΙΟ ΠΑΡΟΥΣΙΑΣΗΣ
ΑΤΟΜΙΚΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Μάιος 2013

Ατομική Διπλωματική Εργασία

RECONSTRUCTION OF EVERYDAY LIFE IN 19TH CENTURY NICOSIA

Καλυψώ Δαυίδ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Reconstruction Of Everyday Life In 19th Century Nicosia

Καλυψώ Δαυίδ

Επιβλέπων Καθηγητής
Γιώργος Χρυσάνθου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2013

Acknowledgments

Firstly I would like to thank my supervisor Dr. Yiorgos Chrysanthou for giving me the opportunity to work with this wonderful project thus expand my knowledge in the field of computer graphics under his aid and guidance over the project. I am also very grateful to PHD candidate research assistant Panayiotis Charalambous, for the precious help he generously offered me, whenever in need or at time of great difficulty. His advice turned out to be a huge contribution to the successful completion of my project.

I would also like to acknowledge and dedicate this thesis to Antonis and my friends Marielli, Julia, Eleni and Christos as they have always been by my side. All of them were constantly encouraging me to work harder, believed in me, relieved me of me of doubts, and set advice to me when needed. They were my inspiration throughout the four years of my studies at the University of Cyprus, and I could always count on them.

I would like to thank my parents and my brother, for bearing with me all the time that I was working on my dissertation and for my absence from home.

Abstract

This project's aim is to extend previous work done in University of Cyprus. This project's overall scope is to present the everyday life of the city of Nicosia during the 19th century, with the different socio-economic identities of its people. The Computer Science Department of University of Cyprus in cooperation with Leventis Municipal Museum of Nicosia, aim to produce a tool which can be used by architects, historians and archaeologists and for educational purposes as well. I had been given models of buildings of Taht el Kale, neighborhood in Nicosia, which were generated using architecture rules and data from land registry from the 19th century. My project's objective, is to augment, this 'cultural site of Nicosia' with virtual civilians with high level behaviour. By inserting realistic civilians wandering around the city, we are giving "life" to the old city. Implemented in Unity 4, intelligent autonomous virtual characters have been produced which look and behave realistically. Each autonomous character is assigned a list of tasks to accomplish such as go to a place, talk with other people, dance, go to the market for shopping, etc. For each character's action the appropriate animation is played and the transition from one action to the other is quite smooth by utilizing Unity's Mecanim animation system. Dynamic tasks are triggered under certain conditions. Steering is managed and collision avoidance between human agents is accomplished using UnitySteer.

Contents

Chapter 1

Introduction

1.1 General

All around us, in any normal day, we can see people moving, waiting for the bus, driving their cars, eating, doing sports, meeting with friends and be part of various activities, many of which, include interacting with each other as well. We often take human activities for granted but if some were missing we would certainly notice. Just imagine a train station or a school without students walking around, working, studying and interacting together. This is the challenge I was faced to accomplish through this dissertation project, by adding to a broader project lifelike virtual civilians behaving naturally in the streets of the city of Nicosia of the 19^{th} century – a town generated with the use of graphics, depicting the city at that era. This project's aim is to present a unique insight to the everyday life of the city of Nicosia of the time, concerning both socio-economic aspect of its people and the spatial organization of the building within the walls during the 19^{th} century.

1.1.1 What is Crowd Simulation?

To begin with, we have to define some terms that will be used throughout this project. The term crowd refers to a group of things or people who are gathered closely together. Crowd may refer to any group of objects adjacent to each other. The members of a crowd may share an image, reputation or even some kind of common behavior.

Crowd simulation is the process of simulating a number of people, masses of creatures or other characters who interact together in the same environment. Each person (called actor, or agent too) has its goals, responds to others and interacts with the environment. Crowd simulation has applications in many places, from architectural planning to improve visually some training environments and virtual realities and to artificially – intelligent characters in movies and games (e.g. Lord of the Rings). Crowd Simulation is frequently used in 3D Computer Graphics and it has many topics to research on, such as real-time path finding, motion planning and collision avoidance. Computer animation and simulation applications can be used for entertainment purposes, education, training and human factors analysis for building evacuation. Many applications simulate scenarios where crowds gather and disperse, work, play, have their goals and some tasks to achieve. These simulations may occur in places like sporting events, concerts or streets of a city.

There are two categories for simulating the movement and the interactions. The first one is called Particle motion and it refers to the motion of point particles, which is animated by simulating forces such as gravity, wind, points of attractions and collisions. This method, although easy to implement, is a bit unrealistic, because of the difficulty of directing individuals and it is generally limited to flat surface. The other one is Crowd Artificial Intelligence, where agents have specific goals and interact with others, as people in real crowds do. Sometimes agents can respond to environmental changes. This way the result is much more realistic than particle motion, but is more difficult to implement. My project is more similar to the last one and it is agent-based, that is, every single one of them has a different behavior and acts as an individual and not as a part of a larger group.

1.2 Motivation

This dissertation project combines both graphics and algorithms, the two subjects I love most in Computer Science field. I am really attracted in everything that has to do with graphics, movies, animation, games, music etc. and that is the main reason for deciding to work with Dr. Yiorgos Chrysanthou as a supervisor. The project's objective, is to add 'human-like' models inside the old walled city of Nicosia and to assign appropriate behavior to them. Being highly interested over animations, I have decided into pursuing a master's degree on specialization on graphics and animation, so as to gain specialization over the matter, thus this project is ideal to work in this filed.

With this project I am also able to work with cultural aspects of Cypriot history and the result of my work can be used for educational and entertainment reasons at the same time. Students can learn through interactive walkthroughs, and at later stager, role playing games within the 3D environment of the city and the everyday activities of the people at the period [1].

1.3 Scope

This project's overall scope is to present the everyday life of the city of Nicosia during the 19th century, with the different socio-economic identities of its people. The Computer Science Department of University of Cyprus in cooperation with Leventis Municipal Museum of Nicosia aim to produce a tool which can be used by architects, historians and archaeologists since they can study the spatial organization of its buildings within the walls of the city. It will be used for educational purposes too, as students will be able to learn the history of the city of Nicosia and everyday activities and customs of people at that time through interactive walkthroughs and role playing games within the 3D environment. This historical era is interesting because in the end of the Ottoman era, Muslims coexisted with the Christian Orthodox population and other minorities, so we can detect diverse roles and civilians' behaviours.

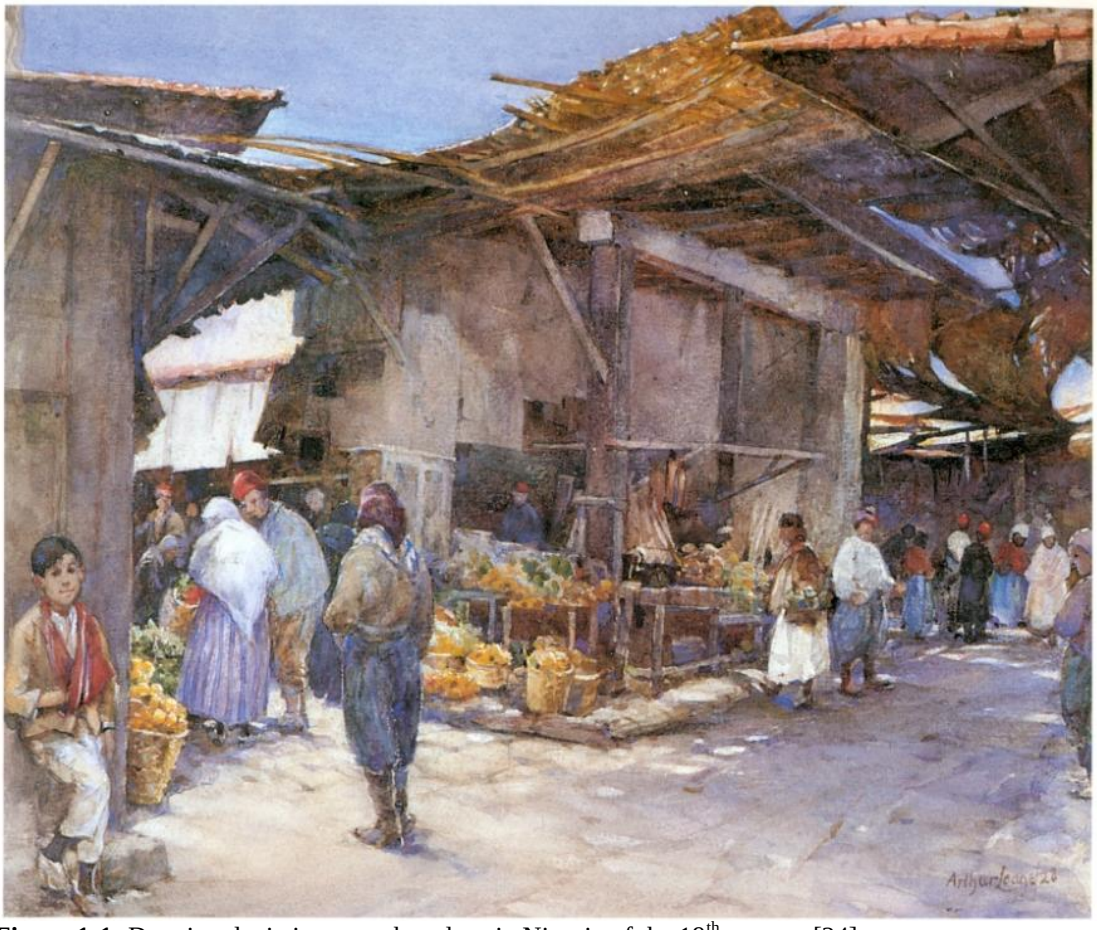

Figure 1-1: Drawing depicting a market place in Nicosia of the 19th century [24]

My project's objective is to augment this 'cultural site of Nicosia' with virtual civilians of high level behaviour. By inserting realistic civilians wandering around the city, we are giving "life" to the old city. So, intelligent autonomous virtual characters should be produced and should look and behave in a realistic manner. Using the information taken from the museum and from Land Registry, old photographs and written documents, the civilians presented in the project will be dressed, act and behave according to their religion, social status and occupation. I developed a simulation, which assigns each autonomous character a list of tasks to accomplish, such as going to a place, talk with other people, dance, go to the market for shopping, take part in social gathering etc… To achieve this, a rule-based motion planning system will be used for collision avoidance. For each character's action the appropriate animation should be played and the transition from one action to the other should be smooth. Low level behavior such as steering is managed, whereas collision avoidance is accomplished using UnitySteer[2].

Chapter 2

Previous work

2.1 Work at UCY

2.1.1 Modeling the Walled City of Nicosia

In 2003 M. Dikaiakou et al.[3] have attempted to model the Chrysaliniotisa Quarter in Nicosia, by creating a partly-automatic system using the GIS data of the region as input to the system and structural analysis of the region's buildings. For this project an official typology of the buildings was needed, as well as the ground plan of this specific area for the 2D visualization. So, the team followed the subsequent method: firstly they proceeded to the construction of the 3D Component Library and the Special Buildings.

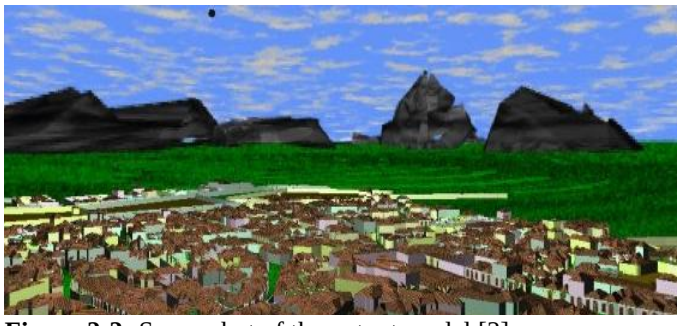

Figure 2-2: Screenshot of the output model [3]

The team has developed some components such as windows, doors, kiosks, arches, roofs and balconies and has tried to combine them in order to create some of the 3D buildings. Secondly the team after reading and parsing the GIS input files, roads are identified and some building features like 'neighboring' and 'facing-road' edges. Houses were classified in four main categories: Original Courtyard, Minimal Courtyard, Planned Serial and New Courtyard. Other buildings were generated automatically with rule-based techniques. Depending on the type of each house, very good 3D models have been produced and the output of the program is realistic.

2.1.2 Reconstruction of Everyday Life in 19th Century Nicosia

My project's aim is to extend the features of the project 'Reconstruction of Everyday Life in 19th century Nicosia'[1] and add lifelike human models to the scene already created. This project involves the recreation of the urban environment of the city based on historic and archival information from Land Registry documentations of the Ottoman era taken by British engineers.

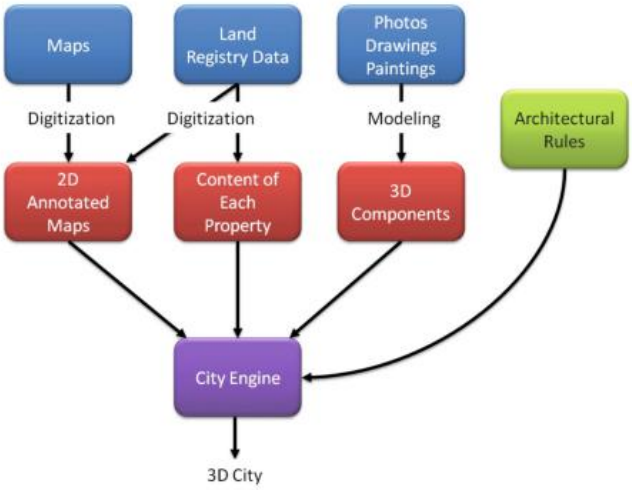

Diagram 2-1: The reconstruction process

Title deed archives of the Land Registry Department in Nicosia, the usage of each room, extra features (trees, wells), the ethnicity and profession of the owner, printed maps, some photos, drawings and paintings as well, were taken into account to create 3D representation of the architectural building types of the walled city. A set of rules and 3D components (windows, doors, arches) were created and passed to a procedural modelling system

Figure 2-3: 3D view of the map in CityEngine

(CityEngine) to generate the city model by creating the 3D model for each house. Rules for different building types (one, two floor house, store) were defined: Structures are first created upon the street line, living spaces took one or two floors, whereas stores rooms, animal sheds, kitchens found to be taking up the back inner spaces[1]. Also all buildings, except shops, are organised around an inner open yard, first floors have smaller windows than upper floors and doorways are centrally set in the facade upon the street. With the process described above the Taht el Kale Mahalla neighbourhood in Nicosia was recreated with the results shown in Fig.2-3 above.

2.2 Crowd Simulation

There are many crowd simulation methodologies and what makes them different is the technique used to generate the motion path. These methodologies are separated in macroscopic and microscopic methods and are used to model the movements of the agents.

2.2.1 Macroscopic approach

Models based on macroscopic approach focus more on the flow - the system as a whole rather than on the characteristics of individual agents and their character and they are used for predicting traffic needs and capacities for large scale structures such as stadiums, airports, train stations etc. In this category fall the next models: Regression models use statistically established relations between flow variables to predict flow operations and flow's characteristics depend on the infrastructure (stairs, corridors) [4]. In Route Choice models characters calculate the in-between destinations of their path, so that they maximize the utility of their trip (travel time, comfort) [5]. Gaskinetics use an analogy of fluid or gas dynamics to describe how crowd density and velocity change over time [6]. Pedestrians in Queuing models move through the network from node to node using Markov chain models by defining a set of states with transition probabilities. Nodes usually are rooms and links are portals or doors [7].

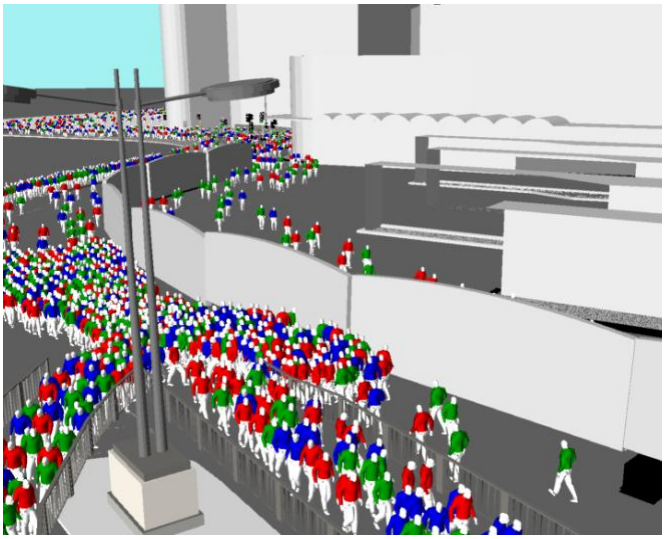

Figure 2-4: Cardiff City match day queue management [25]

Figure 2-5: Agent-based system. People evacuating room [26]

2.2.2 Microscopic approach

Microscopic models study the behavior of the agent, the decisions they make and their interaction with other characters -people models- in the crowd. There are two subcategories of microscopic models: social force models and cellular automata models.

Social Force model is a microscopic, continuous time, continuous space simulation which is using real forces such as friction, dissipation, repulsive interaction and fluctuations to create analogous "virtual" forces. These virtual social forces are used to solve Newton's equation of motion and therefore calculate the motion of each individual character. This model is pretty simple and character behavior is very realistic, it gives the impression of real-world crowd movement. Every character-agent is represented as a small circle in the map and model describes continuous coordinates, velocity and interactions with other objects [8]. Each force is individual for each character and is often chosen based on empirically taken or plausible data. One of the most important models using empirically derived social forces is Helbing's model [9]. To simulate the interaction between people and obstacles repulsion and tangential forces are applied, resulting in realistic behavior. The main drawback, though, of this model is that agents seem to "shake" or "tremble" because of the plentiful "intruding" forces in high-density crowds.

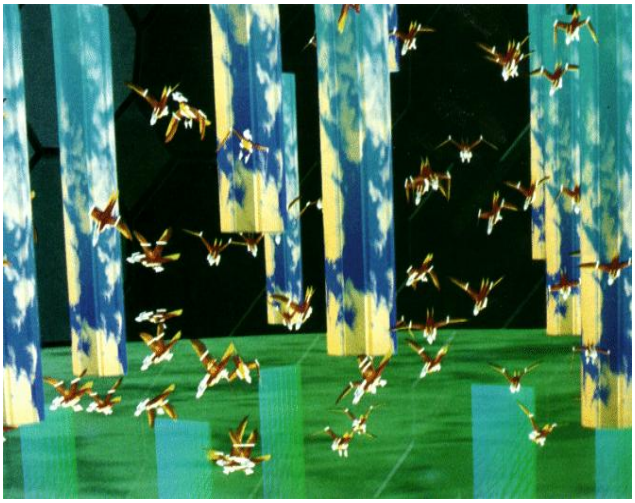

Figure 2-7: Rule based system–microscopic approach [13]

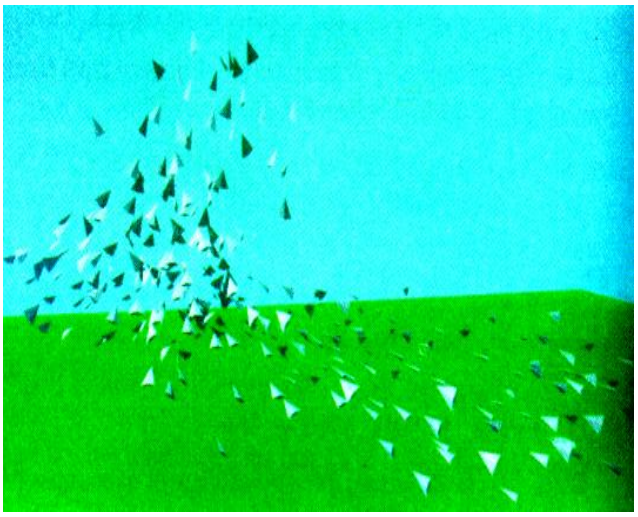

Figure 2-6: Rule based system–microscopic approach [13]

Autonomous Pedestrians[10] is another really interesting decentralized microscopic model which integrates motor, perceptual, behavioral and cognitive components, where pedestrians act as autonomous individuals capable of a broad variety of activities, including selecting an unoccupied seat and sit down, meet with friends and chat and queue at ticketing area and purchase a ticket. This model incorporates appearance, motor, perception, behavior and cognition sub-models. Pedestrians are self-animated and are able to perceive the environment around them (sense ground height, static and mobile objects), make decisions according to them and behave naturally. They also maintain an internal mental state and according to their physiological, psychological and social needs, people with an action selection mechanism choose the appropriate behavior to fulfill the need. A similar example of microscopic approach is the commercial game The SIMS™ 3 [11](Fig.2-8).

Figure 2-8: Left: Autonomous pedestrians wandering in Pennsylvania station in New York. Right: The SIMS 3 game

Cellular Automata Model is an artificial intelligence approach, to simulate crowds where space and time are discrete and physical quantities bearing a finites of discrete values. A cellular automaton is a collection of cells, each in one of a finite number of states, on a regular grid of specified shape that evolves through a number of discrete time steps [12]. The new state of each cell is determined in terms of the current state of the cell and the states of the neighboring cells, in its neighborhood. Floor space is discrete and characters can only move to an adjacent free cell. This results in realistic simulations for low-density crowds, but unrealistic for high-density ones. This approach doesn't allow contact between individuals, although it is fast and simple to implement.

Rule-based models have realistic results for low/medium-density crowds in a flocking style, but do not handle contact between agents, so it doesn't simulate "pushing" behavior. Characters are applying "wait" rules to avoid contact, therefore calculation of collision detection and response isn't needed. A famous rule-based model is Reynolds' "boids"

model[13]. This model is a particle system, where each oriented particle stands for a simulated entity–boid. Every boid is implemented as an independent actor that moves around according to the simulated laws of physics, some animated behaviors and its local perception of the dynamic environment. The simulation of "gathering" behavior is implemented using the rules of 'Separation'-change direction to avoid crowding with other characters, 'Alignment'-change direction towards the average heading of the local agents and 'Cohesion'- steer towards the average position of local characters [8]. The "gathering" behavior requires reaction and data from within a neighborhood in order to avoid collision with other agents and obstacles in the environment. In addition, each agent has access to the whole environment description and boids with this steering behavior are becoming part of a complex autonomous system where they can react with the environment, such reactions are wander, path following, arrival and combined behaviors like queuing, flocking, collision avoidance and leader following.

2.3 Crowd Simulation in Cultural Heritage Environments

My aim is to simulate human agents in the city of Nicosia in the 19th century. Therefore we are simulating crowds in environments of cultural heritage. Various models have been developed to simulate crowds in old towns or markets and are trying to give a glimpse of the lifestyle of the people living there at the era.

2.3.1 Populating Ancient Pompeii with Crowds of Virtual Romans

Similar to our project 'Reconstruction of Everyday Life in 19th Century Nicosia' is the project 'Populating Ancient Pompeii with Crowds of Virtual Romans' [14] whose aim is to present the life of Ancient Pompeii and simulate crowds of Romans and their everyday interactions. The ancient city of Pompeii has been reconstructed with graphics based on archaeological data and virtual Romans were added in the streets to simulate life, just before the eruption of volcano Mount Vesuvius in79 A.D. The models of the buildings were generated with

Figure 2-10: A crowd of Virtual Romans simulated in a reconstructed part of Pompeii [14]

Figure 2-9: the generated Pompeii districts

the CityEngine modelling tool, the same tool used for creating our project's buildings.

Circular areas -graph vertices- show where characters can navigate without colliding with the environment. Agents can move from one vertex to the other, if two vertices intersect. These vertices make up the navigation graph, based on which crowd behaviour is updated individually for each Roman.

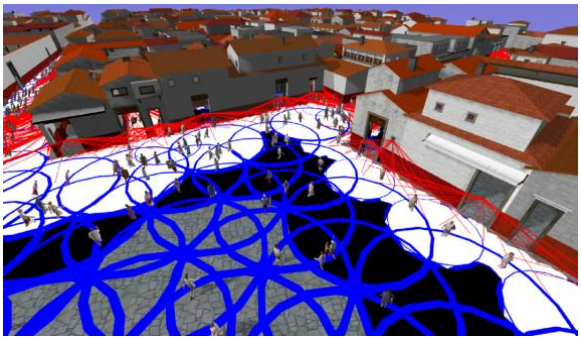

Figure 2-11: Graph vertices are marked with special behaviors: "look at" (in white), "stop looking at" (in black) and "target point" (in red)

To exhibit realistic behaviour of the Roman crowd, two models were used, a high-resolution one for rendering purposes and a low-resolution model to label semantic data on it. The ancient city had some poor and some rich districts and a certain class of people was prohibited to go there, e.g. poor people were restricted in the rich area, so we can only see them wander in poor and older district and vice versa. There are some geometry semantics that trigger specific behaviours and actions of the civilians. When entering a shop, a person walks inside and gets out with amphora or with bread when they enter a bakery. Geometry semantics are placed on doors and window objects too, so that people have to slow down and look through the window or the door.

My project is using junctions to make a graph, whose nodes, each agent will use to find its way through the city streets. Civilians wonder in old town of Nicosia but they are not carrying things around such as bread or vegetables, neither they look towards a door or a window when they pass by. In addition, crowd navigation and path finding is handled using a waypoint graph. Civilians have to fulfil more complex tasks according to their identity. Real data from land registry is used and this way, appropriate behaviour is assigned according to the occupation of the owner of a house.

2.2.3 A Framework for Real-Time Virtual Crowds in Cultural Heritage Environments

G. Ryder, P. Flack and A.M. Day attempted to "revive" Tombland, a section of Norwich [15]. In their implementation, the ground was flat and a combination of methods was used to achieve faster and more flexible rendering resolution. To create the scene in real-time trees were deleted and reintroduced by billboard representations, texture sizes were reduced, walls were flattened leaving a textured polygon, building interiors and back facing walls were

removed and any small objects of insignificant importance were deleted, whereas other geometric objects, such as chimneys, were simplified.

Figure 2-12: A screenshot of the crowd in Tombland, Norwich, UK [15]

Human models were created in 3DStudio Max as Biped models and exported to an OpenGL application and animations were created in Character Studio. For the rendering of the scene techniques like Baked Geometry, Billboards, Color Variety and different levels of detail. Color Variety on the clothes of the human models was produced using GLSL's fragment shader where register combiners based on an alpha mask added color variety to billboards.

To solve agents' placement and navigation Ryder et al. needed a Heightmap, a Spacemap and a Placementmap, which were created by the system. To find where humans can walk and where they cannot a Spacemap was created by testing a bounding box with the maximum dimensions of virtual humans. If this bounding box intersects with any triangle, this sample point cannot be accessed by humans. Placementmap distinguishes space inside and space outside a building and by combining this with the Spacemap we can define if a point is valid starting position for a model or not. So, this project places people within the scene, upon the floor and it generates fast automatic target points within the scene, after the three maps are generated by the system.

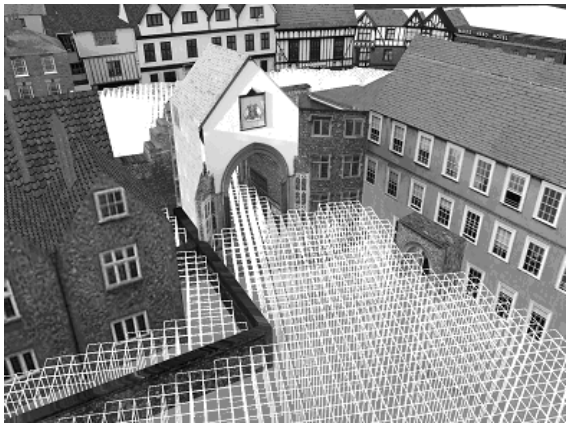

Figure 2-13: Spacemap visualization. Bounding boxes are spaces around the object edges. Some small areas remain inaccessible due to the size and the axis-alignment of the bounding boxes.[15]

Animation was restricted to just walking for simplicity, as the method used lacks run-time flexibility and adds memory overhead to the simulation. The three maps are created offline, once for each model, before starting the simulation for the same reason. Middle level detail mesh (around 3000 triangles per person) was used in order to minimize the rendering cost and achieve better speedup.

My project differentiates, as the civilians in old Nicosia have more complex tasks and a variety of animations thus they are not limited to walking around the city. Cypriots civilians have various animations such as walking, talking, listening, staying idle and dancing. Of course each project is referring to different historic periods and areas. In addition, path finding in my project is implemented with a graph with waypoints, whereas the 'Tombland project' [15] uses bounding boxes to find the accessible areas by testing intersection of them with virtual people. Also in my project I am not using all three maps (Spacemap, Heightmap, Placementmap) and virtual civilians' starting position is statically assigned to each one using data written in a text file.

Chapter 3

Overview

3.1 Project's Components

The overall objective of our project '*Reconstruction of Everyday Life in 19th Century Nicosia*' is to create a realistic model of the town within the walls. My project's aim is to do a small part of the broader project, to add lifelike human models of the town's civilians that behave and act naturally and according to the behavior of the people living in Nicosia at that time, in line with what is written in historical books, photos and drawings depicting the life of that era.

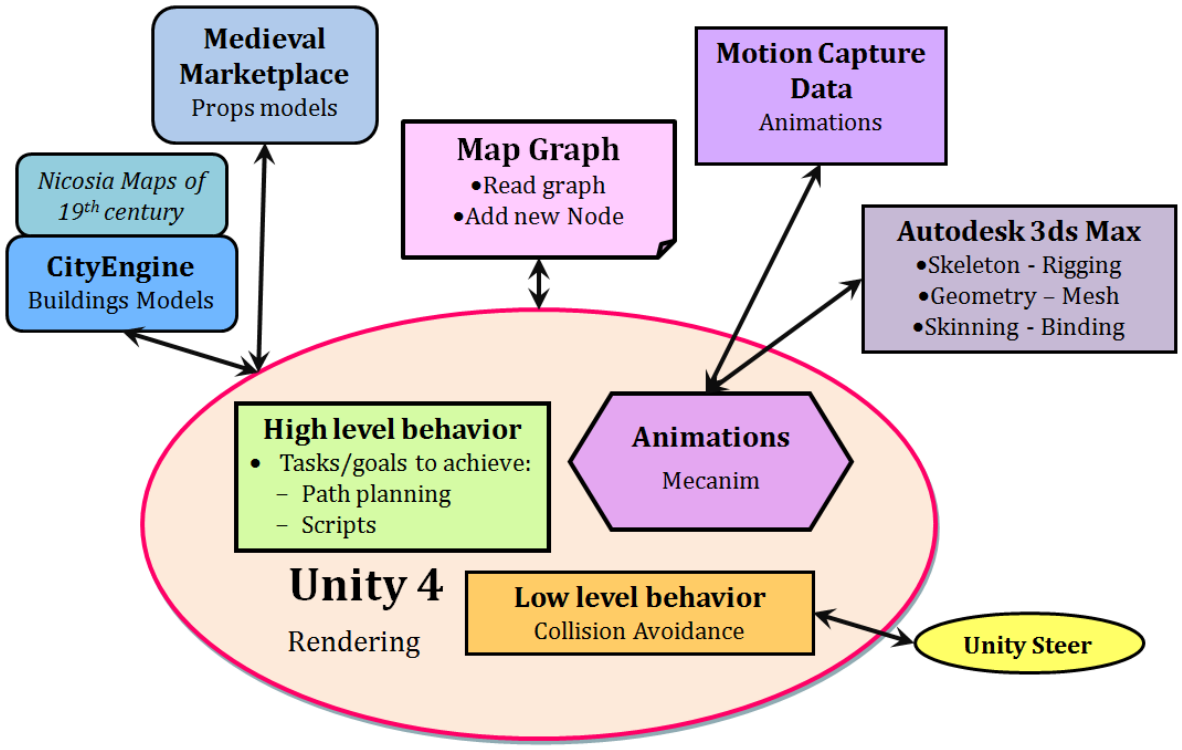

Diagram 3-1: Components are gathered from various sources and imported to this project to achieve a vivid representation of lifestyle

Many features from various sources are required in order to achieve a realistic result. We need models to illustrate the buildings within the walled city and human models to add life to the city. People should have a purpose, some goals to achieve, rather than just wander around the city. For that reason, people should have realistic animations, some kind of high level behavior, low level behavior for collision avoidance and an algorithm for path planning in order to find the shortest path to their destination. Hence, a graph should be generated for the map of the city, so that people can find their way throughout the city. A platform is also needed to load all the components and produce a project that portrays the city along with its civilians, behaving like people would do, if they were living in Nicosia in the 19th century.

We have come up with the following model described in a nutshell below. To create this model we are using work from other projects, previous University of Cyprus students' studies, even models downloaded from Unity asset store (diagram 3-1).

We are using the Unity 4 as a platform for our activities, as it is a powerful rendering engine with a variety of tools, rapid workflows and a user-friendly GUI. Unity[16] is a game

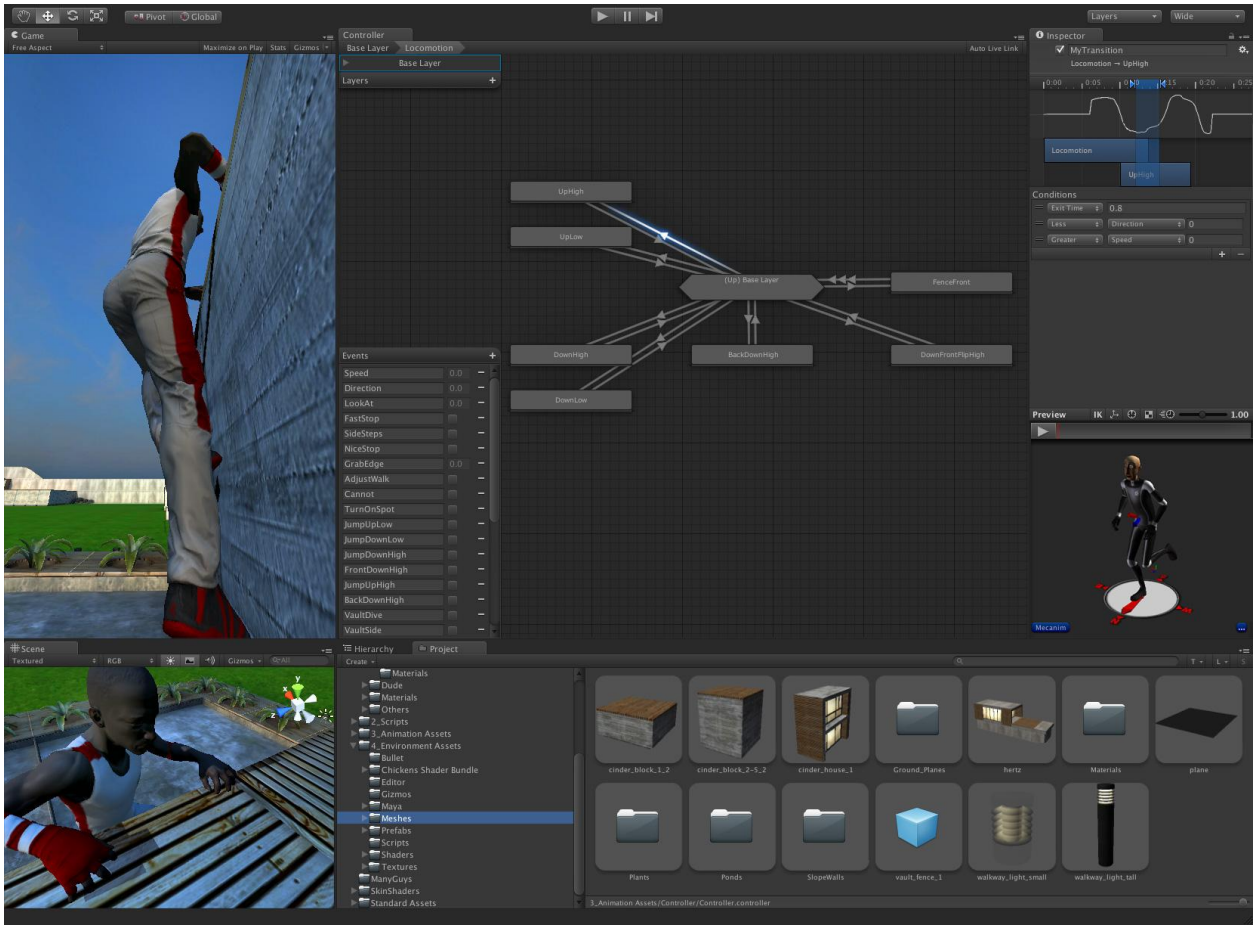

Figure 3-14: Unity 4 interface with Mecanim's animation controller used for blending animations from one to another using a state machine [27]

development system, a cross-platform game engine, that it is widely used for producing video games for desktops, consoles, even mobile devices, supporting development for Android, iOS, Windows, Linux, web browsers, Flash, Playstation 3, Xbox 360 and Wii U. It supports code written in C#, Javascript or Boo. For this project I used C# for my scripts. This game engine handles the rendering of the scene, as it supports bump, reflection and parallax mapping, screen space ambient occlusion and dynamic shadows using Direct3D, OpenGL and OpenGL ES. Scripting is built on Mono, the open-source implementation of the .NET Framework. Unity Asset Store is available within the Unity editor and it consists of hundreds asset packages from textures and materials, particle systems, sound effects to 3D models, animations, even tutorials and ready-made projects. Apart from assets available for download from Asset Store, art assets – models and file formats from 3dsMax, Maya, Softimage, Blender, Adobe Photoshop etc., are supported by Unity and can be easily imported to a project.

Unity's version 4 introduced a new animation system, Mecanim, which blends animations and makes motion of human or non-human characters fluid and natural with an efficient interface. Mecanim is a simple and powerful animation technology, that provides tools for creating state machines, blend trees, IK rigging and automatic retargeting of animations from within the Unity editor.

Title deeds archives provided by the Land Registry department in Nicosia, along with the first ever land survey, conducted in the city of Nicosia headed by Lord Kitchener in the 1880's (Fig.3-15(a)) were used to create a digital map (Fig.3-15(b)) of the city as it was in the

Figure 3-15: (a) Kitchener's Map from 1880 represents an outline of built volumes in the city. (b) 2D digital map of the Taht el Kale mahalla containing architectural ground elevation of buildings with reference to the Land Registry index and information [1]

19th century. These give information not only about the location of roads and houses, but also about how each room in the house was used (kitchen, living space, store room, chicken coop etc), other features like trees, outside ovens and wells, the ethnicity as well as the profession of the owner of each house[1]. Architectural rules of the buildings were created by Charalambos Apostolou, according to this information and fed into CityEngine[1] to generate the models of each building in the area of Taht el Kale. To create the marketplace scene, models from *Medieval Marketplace*[17] were downloaded from Unity's Asset Store.

A graph is made according to the map of Nicosia that was taken from the real data from land registry of that historic era. An input file with the vertices and the edges of the graph is required and this is imported into Unity 4. This graph sets the streets, where people can move and accomplish the tasks given. Also a new node, such as door or shop can be added, even if they were not in the input-file that is used to create the map in the beginning of the program.

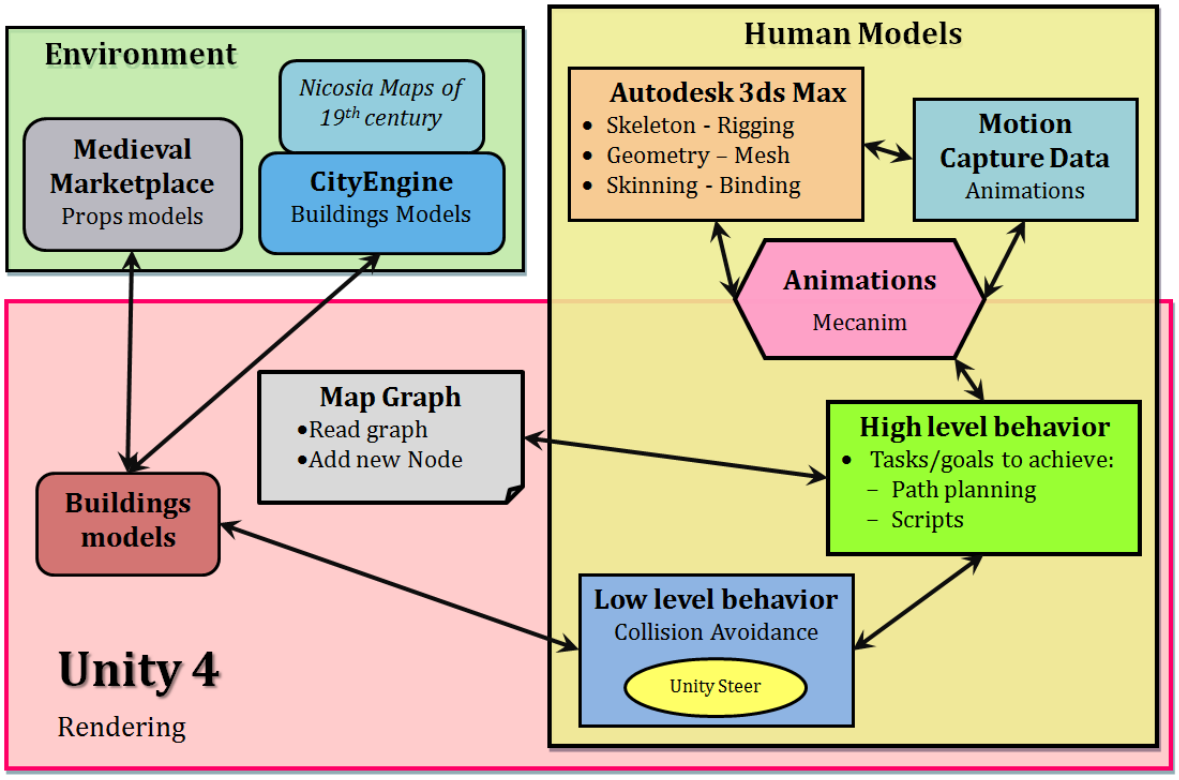

Diagram 3-2: Relationships between the different elements that compose the whole project.

The main components are the human models which are representing the civilians of the old town within the walls. Human models need a skeleton, which is used for animating the character later, geometry, so it can depict a human living in the 19th century in Nicosia and of course skinning, so that the skin-mesh of the character is bound to the bones and therefore the character's movement looks natural. Having the model created in Autodesk 3ds Max as

described above, using Mecanim, Unity's Animation system, we can create smooth animations for our characters. Mecanim uses ready animation clips that were created before, using Motion Capture techniques and makes the transition from one clip to another smooth and natural looking. An animation state machine is created and the transition occurs when under specific conditions, certain parameters are defined, in model's high level behavior. High level behavior determines how the human agent acts. All people have a priority queue of goals to achieve, which contains tasks such as '*move 34'*, *'stay 16'*, *'buy 8'* or even dynamic behavior, as wave to a specific person when they pass by. Collision avoidance is handled using Unity Steer[2] so that agents do not bump into other agents or go through buildings.

Figure 3-16: Pictures and a drawing showing the kind of clothes of people were wearing during that era. These were taken into account in order to create the human models for my project[24].

The various components of this project will be explained in the following chapters with more details. The relations between the various components can be seen in Diag. 3-2.

Chapter 4

Implementation of various modules

4.1 Map Graph

Each person has to find their way throughout the city; hence human models should perform path planning. The city has to be represented in an easy to implement and efficient way, so that human agents can carry out their tasks by finding the shortest path to their destination in order to fulfill all their goals. Our model should define the streets where people can wander according to the real roads within the walled city of Nicosia. After several unsuccessful attempts for path planning, which will be described in the next section, I decided that the best method to use is a waypoint graph, where vertices are the junctions of the roads and therefore edges are the streets where people can walk. Junction is any point in the city map, where it's possible to follow more than one path.

Having taken the 2D digital map of the Taht el Kale neighborhood that was produced by real data from land registry; I have created an input file that describes the graph of this area.

The position of each junction (pos_x, pos_y, pos_z) is the world coordinates, of the real road junction in the Unity project where we have imported the city buildings that were produced from CityEngine. The input file has the format showing in fig. 4-17. Graph section contains the graph's edges, for example line: 7 34 26, means that junction 7 is adjacent to 34 and 26 and there is a road-edge between junction7 and the other two junctions.

```
node_info
<num_of_junction_points>
junID \t pos_x \t pos_y \t pos_z \t JunctionName \t juncType
…
graph
junID \t junIDs_of_adjacent_junctions
…
```

Figure 4-17: Format of mapJunctions.txt file that holds the vertices and edges of the graph representing the roads of the city.

Create Map Graph

The graph of the city's map is read first thing, when the program starts. The input file contains all the junctions, their position, their names and the connections between them. First junctions are added to the graph and afterwards the connections of all nodes are inserted. Junctions can be seen as red spheres on Game mode if *Show Graph on Play* variable, of CreateMapGraph.cs is set to true. Also by setting that to true, connections can be seen in Scene view as red lines. Graph is loaded into a data structure and is used extensively by all agents to find their way through the roads of the picturesque city.

Junction Waypoint Graph

My solution involves a waypoint graph. A graph of this type consists of several waypoints as nodes and the edges between the nodes represent the available roads, the walkable areas. A waypoint – node can be connected with more than one other nodes. An example of a waypoint graph is shown in fig.4-18:

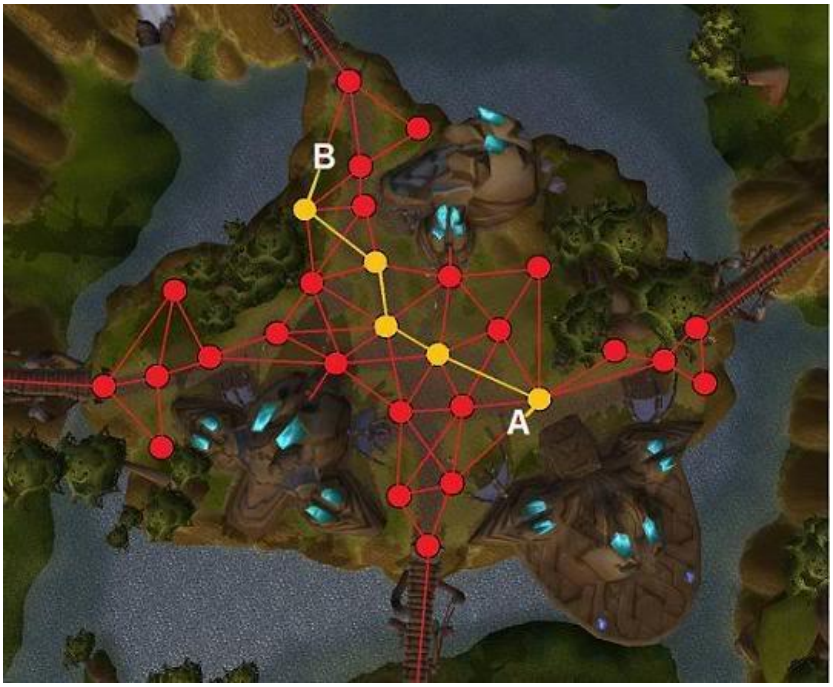

Figure 4-18: A waypoint graph in a scene in World of Warcraft game. Red dots stand for the nodes of the graph, while yellow ones indicate the shortest available path from node A to B. All lines are edges of the graph composing the various paths a kinematic agent could follow[28].

Using *Dijkstra's* algorithm[18] each agent finds the shortest path from junction startID to destID junction by calling function findPath (startID, destID). This function returns an array with the IDs of the junctions that the human agent should follow to get to the destination by the shortest path possible. Hence, the model walks towards the first node, then the second, third… and finally it gets to their destination.

addNewNode_call

We should be able to add new nodes to this graph and update the fixed junctions-those read from the input file. A new node, such as door or shop can be added, if it is not in the input file, by attaching the *addNewNode_call*.cs script to the model that we want to add, as a new node and by selecting from a drop-down list the type of the node.

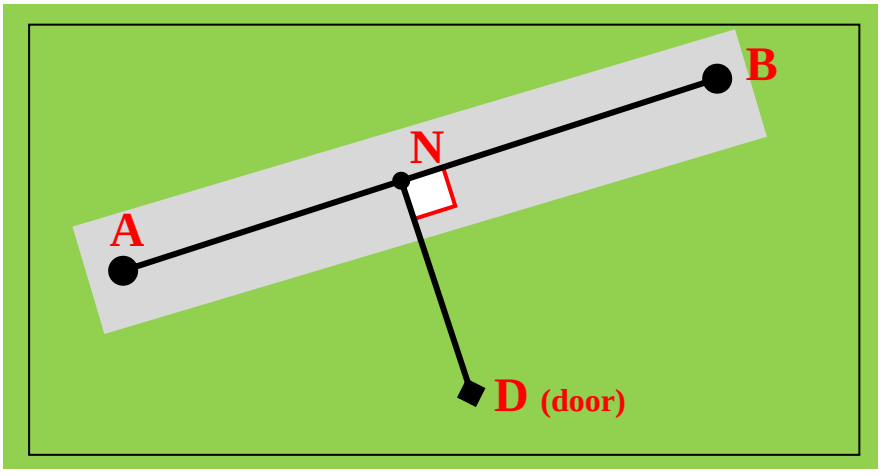

Figure 4-19: Example of the how addNewNode_call works

For example (fig. 4-19), each house's door D should be added as a node to the graph, in order to have people spawning from there. To do so, we find the edge with the smallest distance to this node. Smallest distance from this node is found by calculating distance from point D to all available edges and find the minimum one, the one that is nearest to the point D. To calculate the distance, I have used the distance from a point to a line formula, in Euclidean geometry shown in fig.4-20. Parameters a, b and c are calculated for all edges using algebraic equations.

$$\operatorname{distance}(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}.$$

Figure 4-20: ax+by+c is the equation representing edge AB, whereas (x_0,y_0) is the point D. Although we have 3D space, we are using only X and Z dimensions (instead of X and Y), ignoring Y-axis, the height difference.

Let's say that edge AB (A, B are junctions) is the nearest edge. We add a new node N – in between the two nodes A, B; node N is the point of intersection of edge AB and the perpendicular line to AB, ND. We can calculate the position of N, knowing that ND is perpendicular to AB, by solving the system of linear equations[19] involving the line equations of AB and ND, with the variables elimination method. By using the function addNewJunction (N_pos, D_pos, A, B) we delete the vertices AB, BA and create new vertices NA, NB (AN, BN too) and ND, DN as well. N_pos is the position of point N, D_pos the door position and A and B are the junctionIDs of edge AB, which is the edge with the minimum distance from point D (door). The new junctions added to the graph as well as the new connections created here, can also be seen in Scene View by setting *Show Graph on Play* of CreateMapGraph script to true.

Either by reading from mapJunctions file, or by addNewNode_call execution, if the new node is a shop, its ID is added to the list of shops; if it is a door, it is inserted into the door-list. These two lists are part of the MapGraph data structure.

4.2 Path Planning

4.2.1 Unsuccessful attempts

Human models are wandering in the city and have some tasks to achieve, for example '*buy 34'*, means go to junction 34 – if this is a shop, negotiate with the sales person and then buy something. This indicates that people should be able to find a path to follow which will lead them to their destination, junction34 for example. This path should be the optimum, the shortest available path to the given destination. I have tried several algorithms for path-finding, but unfortunately these did not work for my project. I am going to describe in little detail about these algorithms and then the model of our solution will be explained.

A* pathfinding Unity [20]

I have downloaded 'A* pathfinding project' and I have tried several available methods. In these experiments capsule is representing a human who wants to go to a destination, which in all these cases is the sphere.

Grid Graph [21]

Grid graph method is able to generate nodes in a grid pattern of size width * depth. You have to set to the colliders the Obstacles tag and Ground tag as mask. The properties of the Grid Graph are set in the inspector window as shown in the following figure.

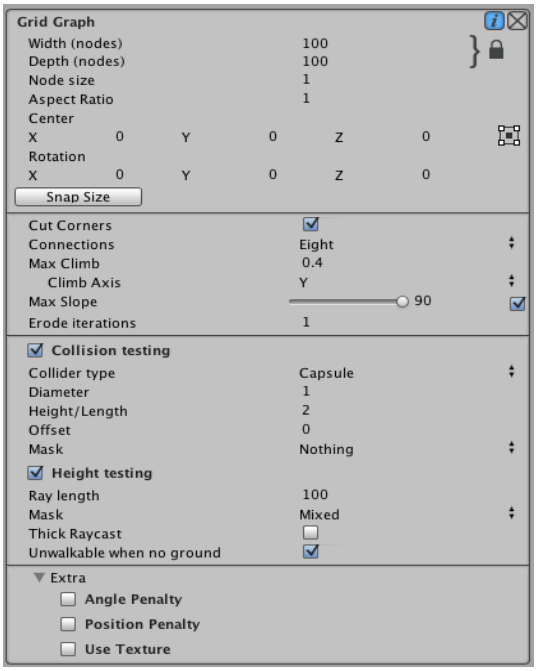

Figure 4-21: Example of settings for the Grid Graph

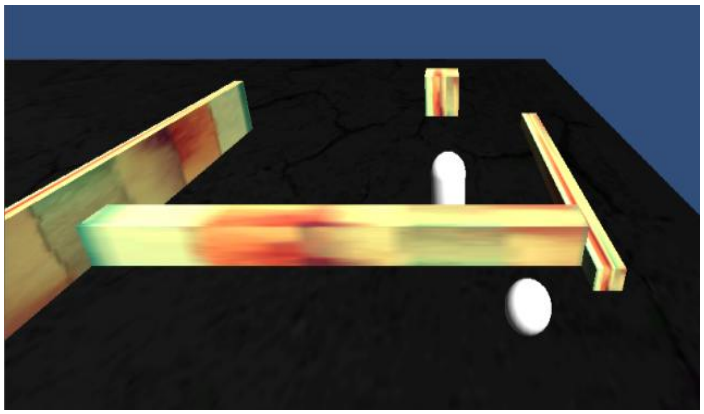

Figure 4-22: Scene on run-time showing that the capsule is stuck behind the walls and can't find its way to the sphere.

The problem I came across here is that the seeker - capsule can't find a way around the wall; therefore it fails in its mission to move to the sphere. Obviously this method cannot be used into my project.

Point Graph [22]

The Point Graph is a simple graph style and it consists of a set of points placed by the user, which are linked together. Every point is treated as a node of the graph and it is checked by raycasting whether two nodes should be linked or not; whether there is an obstacle in-between the two nodes or not. This point graph defines a point of walkability and not an area, something that might end up in not smooth paths. Red cylinders in the following figures

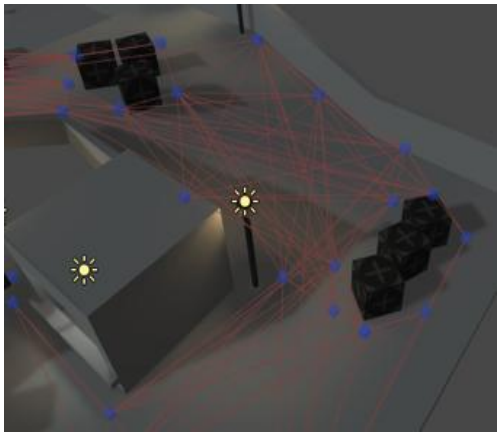

Figure 4-23(a): Example of Point Graph. Notice that there aren't any red lines – connections where obstacles exist.

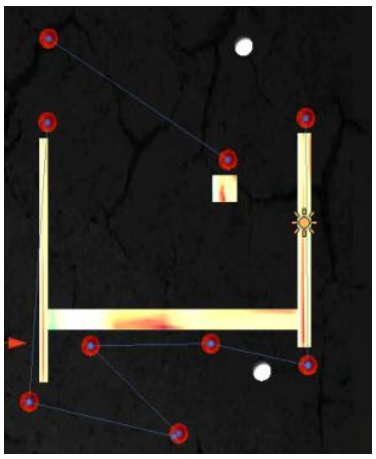

Figure 4-20(b): Points I have assigned for my experiment.

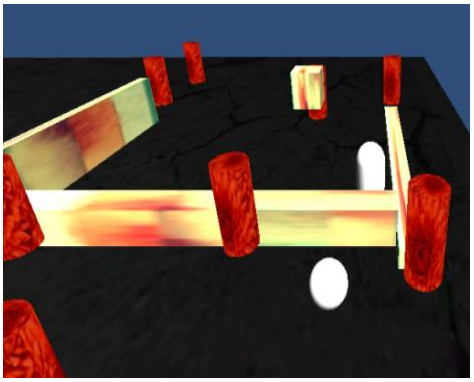

Figure 4-20(c): When executing the program, we face the same problem as the previous experiment.

represent the points of this graph and the blue lines show where a person can walk.

By executing the program we can see that the capsule is trying to approach the sphere, does go near it, but it is stuck behind the wall and can't find a way around the wall to get to the sphere. Hence, this is not an acceptable approach for my project.

Navmesh [23]

The Navmesh Graph expresses the walkable area with triangle mesh instead of squares, as it happens in Grid Graph or with points in Point Graph. It is faster than a grid graph as it contains fewer nodes and it produces a smoother pathfinding.

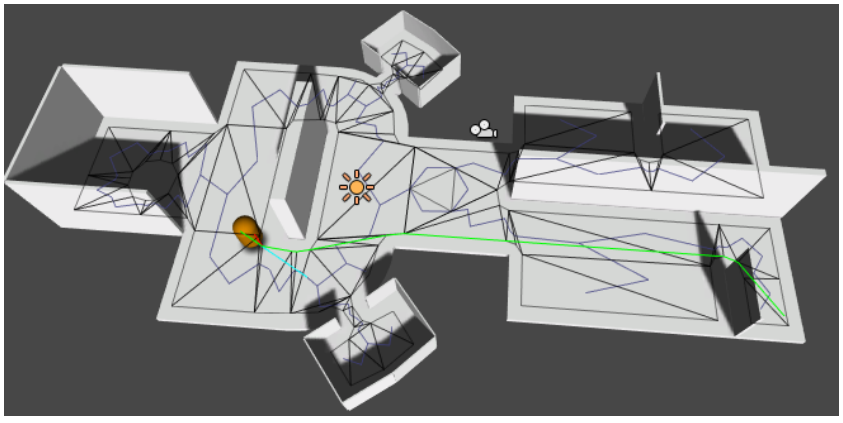

Figure 4-24: A navmesh should be a mesh where polygons describe the walkable area, whereas vertices should lie at the edges of the mesh.

Although Navmesh, it may be appropriate for my project, it needs Unity PRO to generated navmeshes automatically, but since I am using the free version of Unity, unfortunately I am not able to use it. Without

Unity PRO I would have to create the navmeshes for all the city area by myself manually, a very tedious and difficult job. As a result I decided to use a variation of the previous graphs, the junction waypoint graph, which I have previously described.

4.2.2 Junction waypoint Graph

I have created a graph according to the map of Nicosia, the Taht el Kale neighborhood, in relation with the streets of this area. The top view of this region can be seen in the following figure. The red spheres signify the position of the junction waypoints. As I have mentioned before, I have set as waypoints all road junctions, or any spot in the map where a person could take more than one paths. Grey color signifies the existence of streets, where pedestrians can walk.

Each civilian has a number of tasks to accomplish. If the task says *'move 26'*, the human agent must find the shortest path to follow and finally get to destination junction 26. Therefore, the first time a person attempts to accomplish a certain move task, it has to calculate and find the shortest path to follow to the given destination. This shortest path is kept in a stack and the agent starts to move towards the first junction point. When they get close enough to it, the next junction position is popped out of the stack and the agent heads towards this new position. This goes on until all intermediate positions are popped out and the agent should have managed to get to their destination by now.

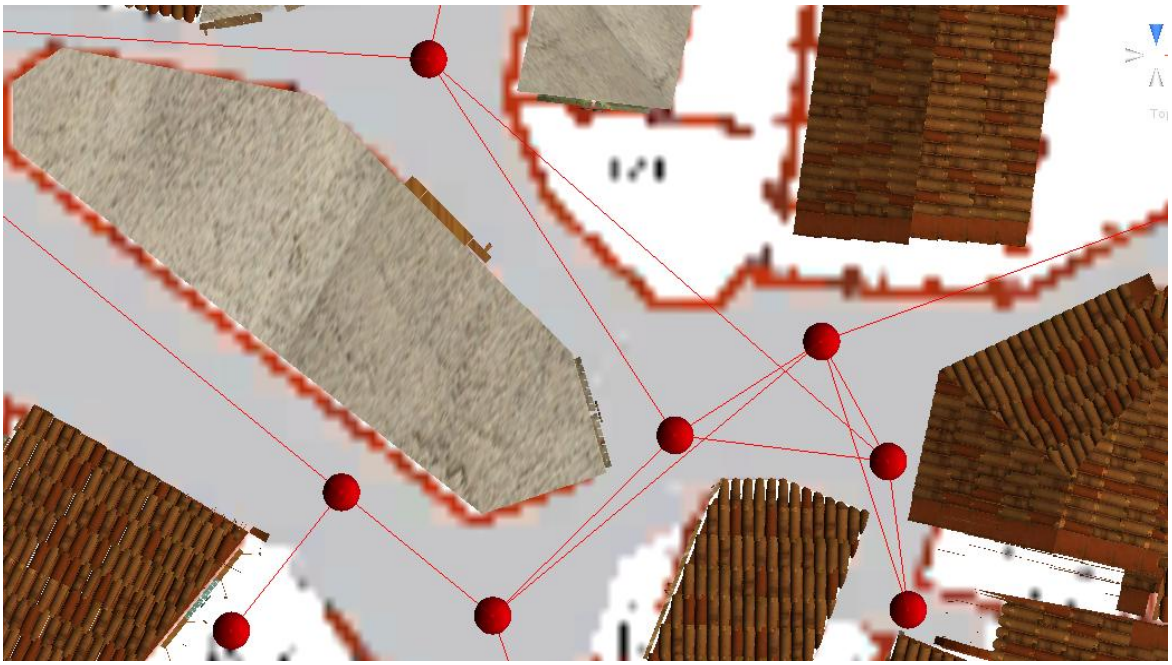

Figure 4-25: Top view of a part of the Taht el Kale neighborhood. Red dots are the positions where a junction waypoint – node of the graph exists. Red lines are showing the edges/connections between those junctions.

In order to find which is the shortest path, each agent uses function findPath(startID, destID) which applies Dijkstra's algorithm, its pseudo code shown below:

```
for each node_junction v in Graph{
        dist[v] = infinity ;      // best distance from starting point to node – v
        previous[v] = -1 ;        // Previous node in optimal path
}
dist[source] = 0 ;
listQ = all nodes;

while ( ! listQ.isEmpty() ){
        u = findNodeWithSmallestDistIn(listQ);
        listQ.remove(u);
        if (dist[u] == infinity)
                break ;       //can't move further from this node

        for each connected_node v of u{
                temp = dist[u] + dist_between(u, v) ;
                if (temp < dist[v]){
                        //new distance calculated is shorter than previous one
                        dist[v] = temp ; //update
                        previous[v] = u ;
                }
        }
}
return dist; //all previous nodes in optimal – shortest path
```

Figure 4-26: Dijkstra's algorithm pseudo code

As long as each citizen can find its way throughout the town, the next step is to assign the appropriate animations according to the agent's behavior.

4.3 High level behavior

Civilians, in my model, have a set of specific tasks to accomplish, rather than just wandering in the streets of the city. These tasks have been already defined in some task files. Hence, when the program is executed, the first thing each agents does, is to read those goals from the appropriate file and load them into a priority queue (implemented by Leslie Sanford, 2006). At the beginning tasks are added to the task list, in a first come – first served manner, as there are not any other tasks at the time. There are several types of tasks, each of them will be explained below, as well as, what the character has to do to get them done. Each task also sets the animator controller parameters, in order to make the agent have the appropriate animation at the given time. The task file can contain in each line only a keyword from the collection *start, move, stay, negotiate, talk, dance, buy, end,* accompanied by a number, separated by a tab – white space. The diagram below demonstrates the inheritance of tasks. Buy, dance, idle,

negotiate and talk tasks, all inherit from Timed_tasks, as all of them happen for a specific amount of time.

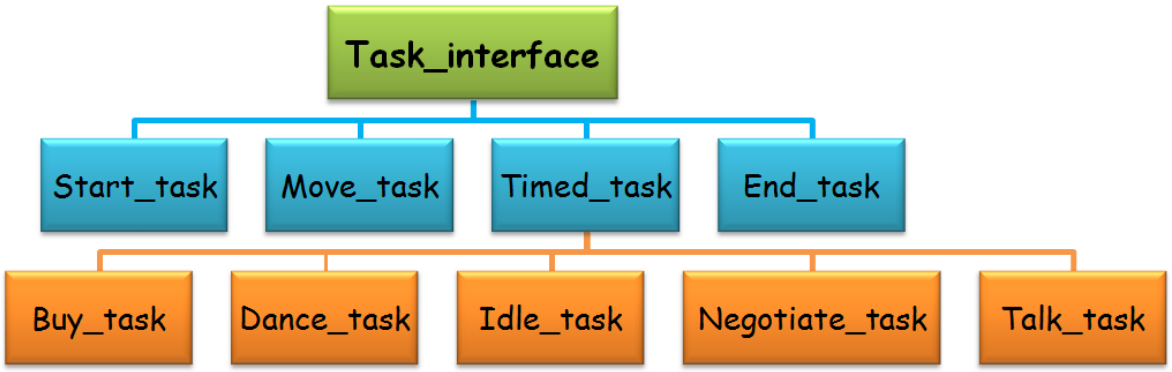

Diagram 4-3: Tasks inheritance model. All tasks inherit from Task_interface and orange tasks inherit from Timed task

If instead of a number, integer or float, symbol '#' is written in the input task file, then this stands for a random number, either junctionID or number of seconds this action is going to last.

Civilian on FixedUpdate(), which is called every fixed framerate frame, checks if the task that is trying to achieve is completed or not yet. If it is not done yet, then doTask() is called in order to complete the task. If the current task is already accomplished, the nextTask is popped out of the priority queue and agent tries to fulfill it.

Start Task

This task is always at the beginning of the task list, so it always happens first. The number that accompanies 'start' keyword represents the id of the junction where the character should start its journey from. According to the norm, people would start from a house, therefore the position of a door is a smart choice. If there are currently no doors in the door-list, agent will begin from a junction randomly selected. Else, if the symbol '#' goes with *start* keyword then a doorID from the door-list is selected as the starting point for the character. If a number is given, then the character starts from that junctionID.

Move Task

Move task is the task that gets the character moving around the city, wander in the streets and go to the market and it is invoked with the '*move'* keyword. In case of symbol '#' a junction is selected in random to go to. If the number given doesn't correspond to a node of the graph, then this move task is ignored and agent moves on to the next task.

The first time doTask() of move task is called, the path that the character should follow is not calculated yet and that's the reason that getsPath(startID, finalDestID) is called. This function stores all the in-between nodes of the graph that the character should follow in order to follow the shortest path available to get to the destination nodeID. The calculation of the shortest path is done as described before in section 4.2.2.

If the difference of the destination position and current position is larger than a threshold, meaning that that the civilian has not successfully moved to the destination node yet, then target point of steerForPoint is set to be the destination point and the autonomousVehicle script of Unity steer makes translates the model's position towards it. At the same time walkState parameter of the animator controller is set to true. This switches the character to the walk(inplace) animation and as a result the model moves its legs like walking and its position is changing and it looks like its walking in a natural way. The speed of walking can be defined in the Inspector when the DancerScript.cs is attached to the human model.

If the difference is less than or equal to the threshold, then the next available in-between node is popped out. If there are no nodes left and the difference is smaller than the threshold, the agent has finally moved to their destination and the walkState parameter is set to false so that it stops the walking animation. When character has reached a node and has to move on to the next one according to the path calculated the model had to turn towards the next node junction. This should be done smoothly in order to look natural and not rushed or forced. Therefore I have handled steering with the method described in section 4.5 – Low-level behavior. This method involves turning slowly towards the next destination when the civilian gets close enough to an in-between junction.

End Task

End task is the last task in the character's list of tasks, invoked with the '*end'* keyword. This keyword should be the last task written in every civilian's task file. The character sets animator controller's danceState parameter to true, and this is the animation that they have until user decides to finish the execution of the program.

Timed Tasks:

Timed tasks require a kind of timer in order to check the duration of each state. For this reason, the first time each timed task is called, the start time is stored and in every frame the difference of current time and start time is checked, to calculate if it matches animation's duration.

Buy Task

The keyword '*buy*' signifies that the civilian should move to the shop specified, negotiate for a little time with the salesperson and then buy something from the store. In the case of no shops in the map, this task is ignored. If the ID specified does not belong to a shop or symbol '#' is used, a shop from the list is randomly selected. Afterwards 3 tasks are added to the task list: a move task with the shop as set destination, a negotiate one with fixed duration and a buy task, which sets the negotiateState to true, in order to switch to the Negotiate sub-state machine and then sets the grabState to true, to apply the appropriate animation, which causes the agent to reach the salesman, to collect the goods bought. By the time it finishes shopping, negotiateState is set to false, and it moves on to the next task in the list.

Dance

This is another timed task, which has duration in seconds the number that comes with '*dance*' keyword. If in the task input file symbol '#' is written, a random number in the range of 5 and 20 seconds is selected as the duration of the dance action. In the animator controller danceState is set to true and only becomes false, when the duration of the action has passed.

Idle Task

Idle task is invoked with '*stay'* keyword which is a timed task with duration of the number set. If instead of number, '#' is given, then duration is a random number between 3 and 30 seconds. Because there are 2 different idle states, when the task is set, idle1 or idle2 is selected in a random manner.

Negotiate Task

Negotiate task, is a task that can be divided into two smaller tasks. Symbol '#' signifies a duration of greater than 5 and less than 15 seconds. negotiateState parameter is set to true and talk1 or talk2 state is set to true in the first half of task's duration. In the second half talkState is set to false and listenState is set to true. This way, the human model first talks for half the duration and listens the remaining time. Therefore, we create an illusion of interaction between a potential customer and a salesman. When the timer meets duration time, listen, talk and negotiate states are all set to false.

Talk Task

As all the previous timed tasks, talk task has the duration of the given number or a number between 4 and 20 seconds. It is invoked by '*talk*' keyword and it sets negotiateState and talkState to true when the task begins to be executed and both these parameters to false when the task is done.

4.3.1 Triggers

Triggers are some actions that happen only when something enters a characters Collider. Usually characters have a Sphere or Capsule collider component attached and when another collider enters this area, function OnTriggerEnter() is called. OnTriggerExit() is called respectively when someone is leaving the area of the Collider.

addNewTask

Some people have the addNewTask script attached. When a character's collider intersects with another person's collider, OnTriggerEnter() of DancerScript.cs is called, character stops whatever they were doing and a new task is added in their list with the highest priority-0. This happens only if the person, whose collider has collided, has the addNewTask script

attach and has the same type of Task too. That means that when two people who both have a dance type task defined in their addNewTask.cs script, when they get close enough, they will turn to look at each other and start dancing together. Likewise, when people have talk type Task, they will turn to look at each other and start talking. The addNewTask.cs script is used to define the type of the new task in Unity's Inspector from a drop-down list. Furthermore, steer for Neighbor Avoidance script is disabled, whereas steer for Point's target is set to models current position, in order to stop moving and not avoid the other person. The task that was interrupted before is resumed, that is, is continued from where it was left, just because it had been re-entered with priority-1 in the list and therefore it will be executed right after the new task is done. For instance, if the character was in walking state, character continues to follow the path to their destination. If a timed task was interrupted, then after the new task, the interrupted one is executed, with the remaining time as duration.

salesPersonTrigger

Sales person, is the person behind a kiosk or anywhere in the market who is selling something, fish, bread, meat or fruits and vegetables. These people have the salesPersonTrigger.cs attached in order to interact with the possible customers that come to their desk. When a character gets close enough, inside the salesman's collider, salesman starts to talk. As the customer is negotiating first they will talk and then they will listen to the salesperson. That's what the salesPersonTrigger.cs does. It ensures that when customer is in talking state, salesman is in listening mode and vise versa. When buyer reaches out to get the goods, salesman also reaches to hand out what the customer bought. Also it makes the buyer to turn and look at the salesman when they talk to him.

waveTrigger

Wave trigger is invoked when two wandering people, not salesmen, are close enough and as a result the two people are waving when they pass. This happens when. waveState parameter is set to true in OnTriggerEnter() and is set to false in OnTriggerExit().

4.4 Animations

My project's objective is to add life-like human models in the city of 19th century Nicosia. But, what would it take to make the city come to life? Civilians should have the appropriate animations according to their behavior in order to produce a realistic illustration of the life in that era. Several animations correspond to different actions a person can perform.

To handle the animations of the human models I have used Unity's Mecanim Animation System which provides an easy workflow and setup of animations on humanoid characters. Once the animation imported, it can be applied from one character model onto another, as long as both characters are set up as Humanoid characters. Jumping from one animation to the next one, would have been a very difficult job to do, nevertheless by exploiting the advantages of Mecanim, this can be a simple task since Mecanim provides a visual programming tool to manage complex interactions between animations. It is really important to have a smooth transition from one animation to the other, otherwise the movement of the character looks unnatural and not human-like.

For this project I have used Animation State machines. Every character is always busy performing a particular action at any given time. These actions are represented as different states in the State machine that is different actions a human model can perform. One of the states is set up as default, shown with orange color and the next state depends on various parameters. Depending on the current state of the model and the parameters at a given time, it can switch to different states. This transition from one animation onto another are defined with arrows from one animation to the next one. Each of these transitions has certain restrictions defined by user. For example, a model can switch from idle state to walk, only when parameter WALK is set to true. This parameter can be set through programming, for instance when user pushes a button, the WALK parameter is set to true and character switches to walking state and performs the corresponding walking animation. Parameters can be of vector, integer, float or boolean data types and a transition can be defined to happen after a certain amount of time or if a parameter is less, greater than or equal to a specific number.

I have created an animator controller for all civilians that are wandering around the city and another one for salespeople, the people who are in the market selling something. There are also two variations of the salesperson animator, with different default state each, just so they don't start all with the same exact animation, which will not look very natural.

Salesman1

A salesperson is a non-moving character so its animator does not include a walk animation. There are several idle animation states and it switches from one to the other, when a certain time passes. When a person gets close enough to the salesperson to buy something, it triggers the salesperson to switch to Negotiate state, who starts to talk, call out about the goods they

sell. Salespeople can talk to and listen to their customers and perform the grab animation, in order to hand out the goods to the person opposite them.

Each transition, as it can be seen in the figure below (c), can be modified so that the transition from the one state-animation to the next lasts longer or shorter amount of time and user specifies the exact time interval when the transition starts and ends. I have modified all transitions in order to have smooth and natural shifts.

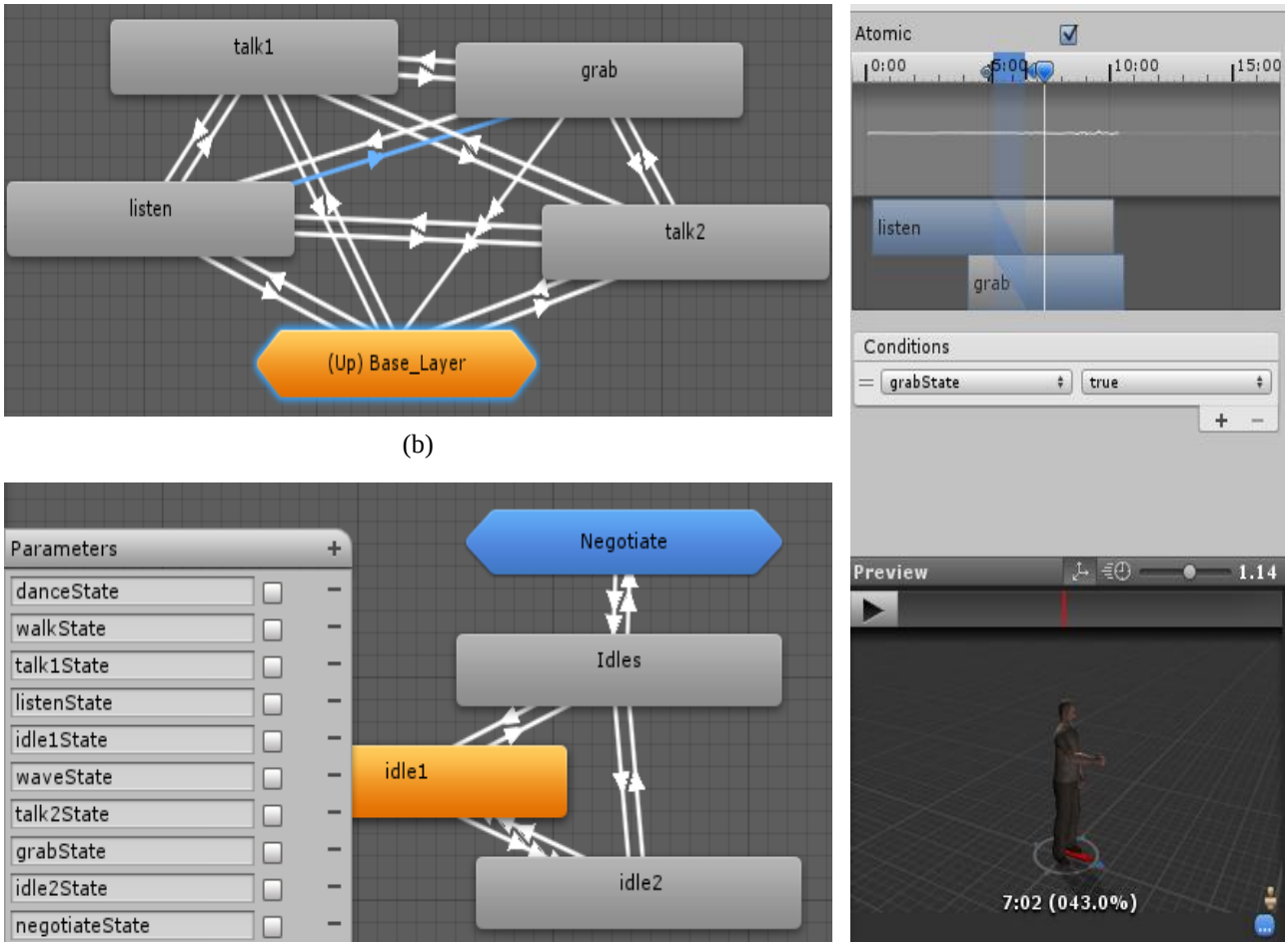

Figure 4-28: Base animator layer of a salesperson. Different idle states in picture (a); salesman switches to Negotiate state when a person approaches. Negotiate sub-state machine (b) is composed of talk1, talk2, listen and grab animations. Selected transition in (b) can be modified in (c) by choosing when and how much to blend the two animations.

CypriotAnim

All people, not salespersons have this animator attached to them. Base layer (Fig. 4-29) is composed with idle1, idle2, dance, walk and negotiate states.

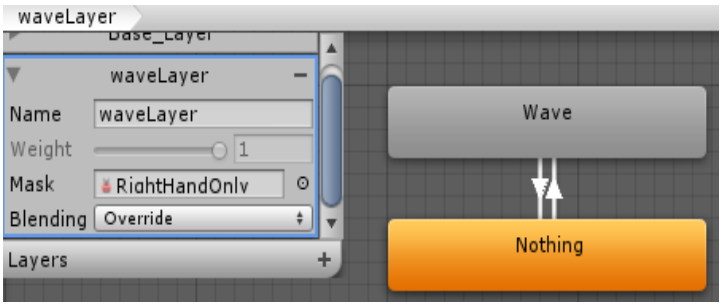

Figure 4-27: Second animator layer of Cypriot person, gives them the ability to wave at other people.

Apart from idle states the model can move around the city to buy something or talk to someone when they pass by. It can even dance. Negotiate state is pretty much the same as the negotiate state of the salesperson. Agent can talk, listen to the salesman and buy something by switching to grab animation – state. Transitions are switched the same way as a salesperson does, that is, when the specific parameters do change.

People with this animator, utilize the waveLayer (Fig. 4-28). Base layer is applied to the full body of the character, which means that the whole body is animated, while wave layer only applies to the right hand of the model. This is achieved by applying a custom mask, which involves only the skeleton and muscles of the right arm of the model. When two people are close enough, their colliders intersect, they wave at each other. Because the blending is set up as override, the wave animation will override the animation of the right arm only, whatever it was doing at the time. When for instance, a character passes by another, while walking to his destination, his right arm waves, without affecting the walking animation of the rest of his body.

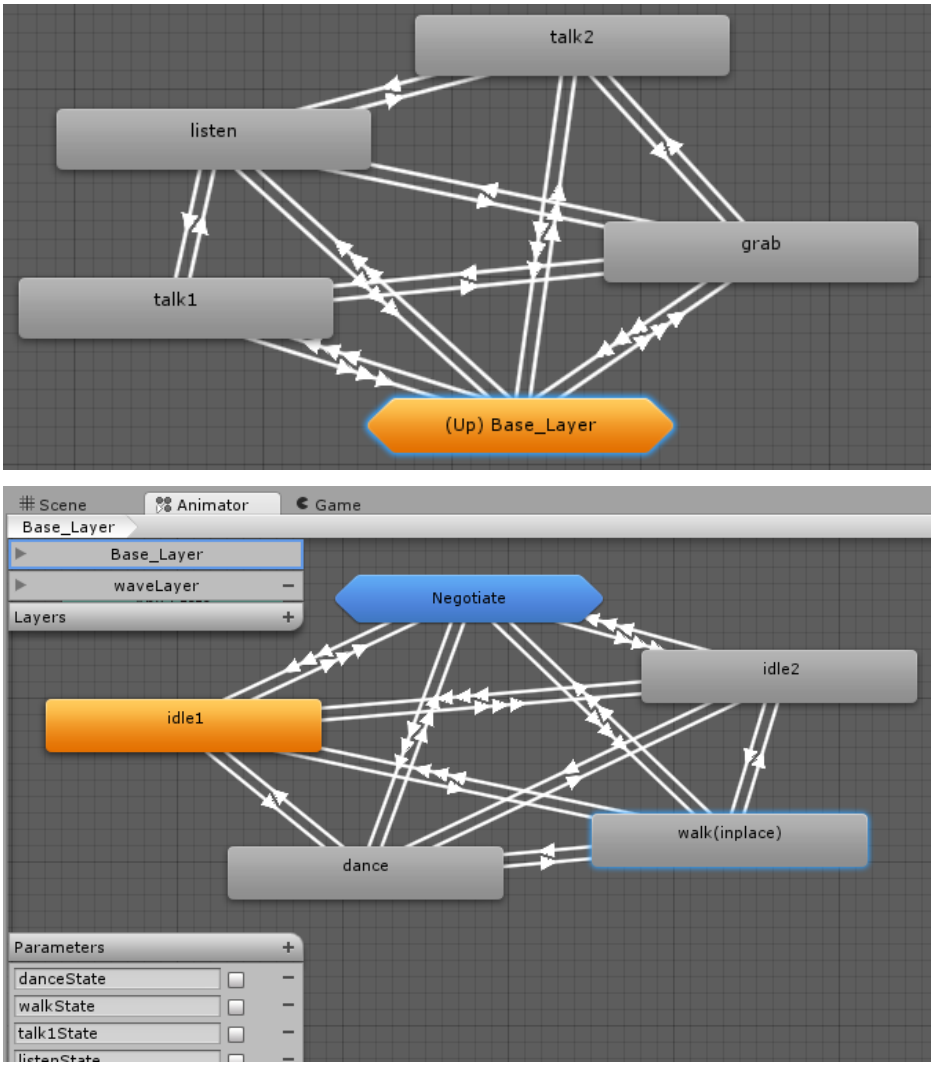

Figure 4-29: Cypriot controller, all people except sales people have this attached to their Animator controller

4.5 Low level behavior

4.5.1 Steering

Citizens in order to fulfill all their goals have to wander around the city and through the narrow streets. Both *'move'* and *'buy'* keywords invoke tasks that make the character move and follow a certain path. As a character follows a path, first they look at the first node that the path contains. Then, for example when a man gets close enough to the node he starts turning gradually towards the position of the next destination. If we don't handle the turning of the agent in each corner, the character would move to the first node and then would immediately turn and look at the next node position. This movement would look forced and of course not natural at all. Therefore I have tried to solve this problem, by using the method described below.

Steering is managed during move task, as it is the only kind of task that the character has to follow a path. Move task is described previously in section 4.3. The following figure explains how steering works:

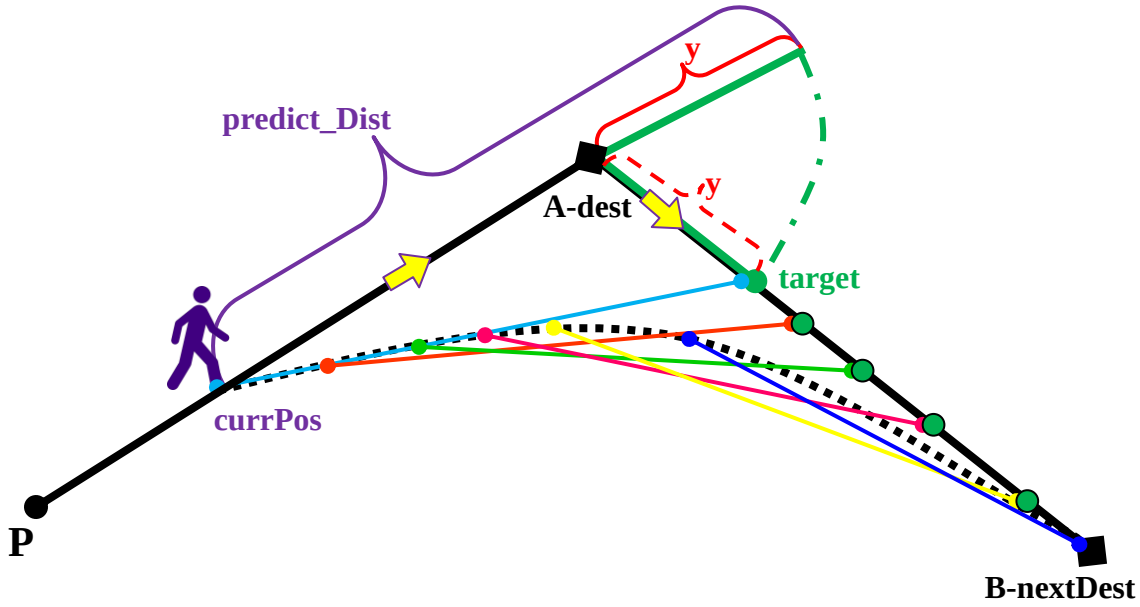

Figure 4-30: Steering example: The character is currently at *currPos*. The path to be followed order to move first to A-*dest* and afterwards to B-*nextDest.*

Let's say that character will move first to position A-dest and then to B-nextDest. For convenience we are going to use terms A and B. There are two cases: B is the last node, the final destination of the path and B is an in-between node of the path – not the last one.

In the first case, where B is the final destination, character should move to A and gradually start turning towards B, though, agent should just go to B and avoid making calculations towards the next turn-destination, just because there is no next destination. When a destination is the final one, nextDestID is set to -1 as flag. So, if the distance between current position and final destination – B is larger than a threshold, meaning that character hasn't reached destination yet, agent should look towards this final position and move towards it. If this distance is less than threshold (0.5) agent stops walking animation, sets move task to done and stops moving forward. If next destination is the position of a shop, character moves towards the target calculated (not shop position), as salesPersonTrigger of salesmen assures that buyer looks at the salesman when buyer enter their collider, consequently this will make the buyer to look at the salesman as the buyer approaches.

In the second case, turning should continue after A, towards B and afterwards towards C, the next node position of the path. To calculate the rotation on y–axis of the character, in each frame character looks at a calculated target position only if *predict_dist* is greater than *(|dest–currPos|.magnitute).* Otherwise, agent looks towards A (dest position). Target is calculated by:

$$target = target + (y * |\boldsymbol{nextDest} - \boldsymbol{dest}|.normalized)$$
$$y = predict\_dist - |\boldsymbol{dest} - \boldsymbol{currPos}|.magnitute$$
$$predict\_dist = walkSpeed * LOOK\_AT\_TIME$$

Figure 4-31: Target position calculation

LOOK_AT_TIME = used to calculate how far ahead the character looks

| dest – currPos |.magnitute = distance between dest and current position

|nextDest-dest|.normalized = normalized vector between dest and nextDest

predict_dist value depends on LOOK_AT_TIME. This time variable defines how "early", how far away will the character walk, until they reach a point where character starts to turn towards the next destination in their path.

$$\left(\frac{\boldsymbol{v1} \cdot \boldsymbol{v2}}{||\boldsymbol{v1}|| * ||\boldsymbol{v2}||} > thresh\right) OR$$
$$(|\boldsymbol{nextDest} - \boldsymbol{currPos}|.magnitute < distThr$$

Figure 4-32: Condition to stop turning smoothly and move on to the next node of the path: Cosine of the angle between v1 and v2 OR character is too close to nextDest

v1 = |nextDest - dest| vector

v2 = |nextDest - currPos| vector

v1 · v2 is the dot product between v1 and v2

$||\mathbf{v_i}||$ = is the magnitude of v_i

Citizen turns in every frame until vectors v1 and v2 are almost the same. Therefore thresh should be as close to 1 as possible ($\cos(0^o) = 1$). Also, if character is very close to B (nextDest), character should move on and set dest and nextDest accordingly. In case that the inequality in Fig. 4-32 is true, agent has already turned towards dest or they have reached nextDest. Hence dest takes the value popped node (= nextDest) and nextDest takes the value-position of the node (after pop()) on top of the stack which contains the path. Target now takes the value of dest and it will not change until predict_Dist becomes greater than *(|dest–currPos|.magnitute)*. If there are no nodes left in stack, we set nextDestID to -1 in order to stop turning when dest is the final node of the path.

In the following figure, several frames were taken at the time that a human model was executing a walk task. Frames are sorted earlier on the left and the later ones to right, showing how the human model is rotating little by little unto the next node position. Notice that character does not wait to reach the first destination (red sphere) in order to start rotating (turning) and as a result a smooth rotation is observed.

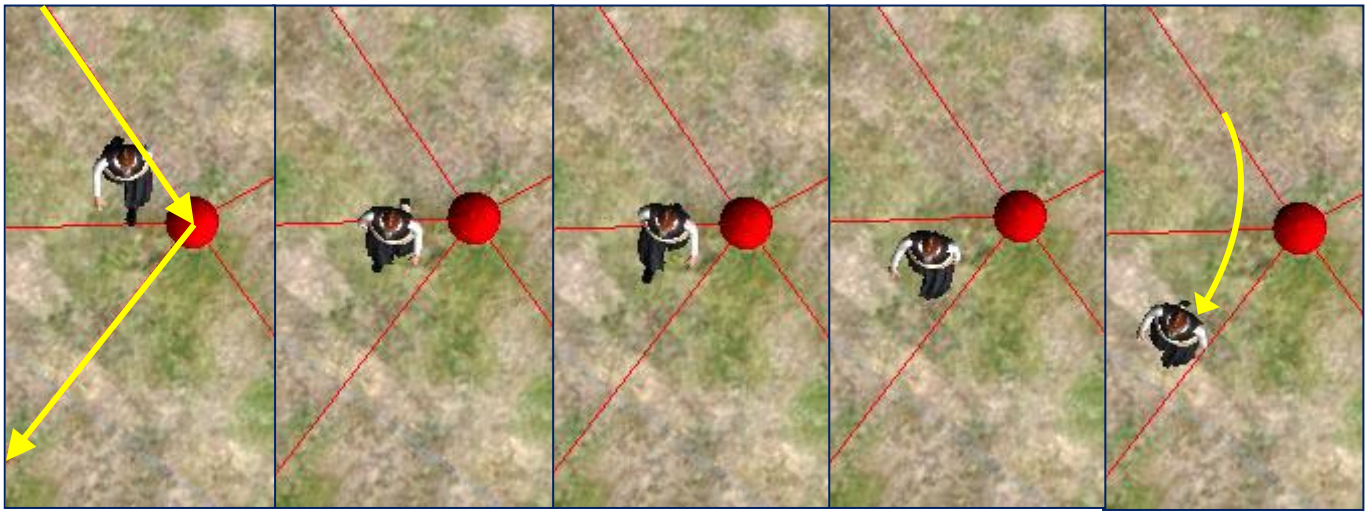

Figure 4-33: Cypriot character turning gradually towards the next destination, showing the results of steering

4.5.2 Collision avoidance

Collision avoidance is the process of preventing something from colliding with another object. In crowd simulation, collision avoidance means preventing human agents from colliding with other humans, as well as obstacles, such as buildings or scene props. In my project I have successfully applied UnitySteer' s code to my models in order to avoid colliding each other when walking. UnitySteer[2] is a library of steering components for Unity based on OpenSteer, covering anything from path following to obstacle avoidance to

following neighboring vehicles. The library is under the MIT license and hosted on Github. Unfortunately, I could not manage to make human characters to avoid collision with buildings.

In order to avoid walking into each other we set the layer of every single wandering person to be *Neibhbors* and we attach the following scripts of UnitySteer:

- Autonomous Vehicle
- Radar Ping
- Steer For Neighbor Avoidance
- Steer For Point

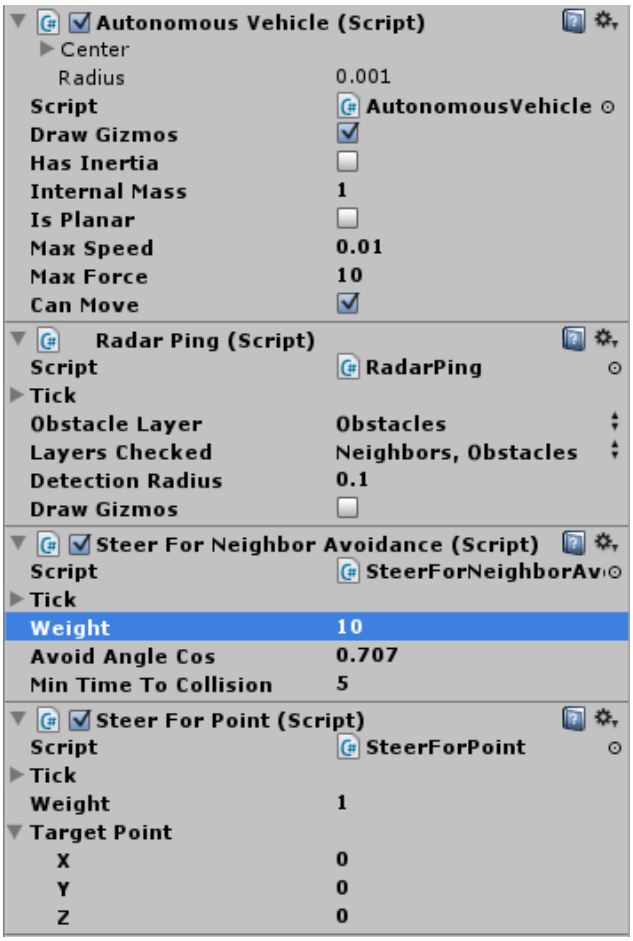

Figure 4-34: Scripts necessary for collision avoidance between human agents.

In Autonomous Vehicle we set the radius according to the human model size and the max speed that we wish our civilians to have and as a result this script makes the character move. Radar Ping is the script that checks if another game object of Layers checked is inside Detection Radius. Therefore we set it to check for *Neighbors* and *Obstacles* layers and the detection radius large enough so that we can detect people who are close and there is a possibility to collide with them. Steer for Neighbor Avoidance is the most important script for collision avoidance, where we set how much weight we want, the rotation angle with which a person will avoid another and how much time before collision we want our characters to change direction in order to not bump into each other. Now, Steer for Point works along with Autonomous Vehicle. to move the character to a certain position. This position is the Target Point.

In general by attaching these scripts to a citizen model it makes it check, for other citizens less than a certain radius away and if it predicted (according to their speed and direction) that they will collide, both citizens rotate a small angle and as a result they avoid crashing into each other.

When an agent starts their journey, when executing *start* task we disable both *SteerForPoint* and *SteerForNeighborAvoidance* as we only need to avoid people when walking. Thus in the beginning of every *move* task, we enable them both, in order to avoid other wandering people. Of course when a person reaches the end of their journey both scripts are disabled again so that people do not move when in idle, talk, dance, etc. mode to avoid other people that are approaching.

Knowing the importance of collision avoidance with nonmoving obstacles, this is something that I tried hard on achieving. Unfortunately, due to lack of time and the rate of difficulty of the process, I did not succeed in implementing it successfully. Thinking creatively I made an attempt to achieve my goal by using UnitySteer, which provides code for avoiding collision with spherical objects. Having this code people check if there are any spherical obstacles close to them and calculate the force needed to steer to avoid the closest ones. I have tried to find circle – rectangle intersection, but I was not successful in my attempt. Although I have managed to find point – rectangle intersection, having known whether an obstacle is colliding with our character or not, still I had difficulty in calculating the force needed to avoid crashing into the building, thus having an end result where the agent, ends up walking through the obstacle. I would like to register this, as drawback in my project, as still is not implemented, and collision avoidance for the buildings inside the city, is still a task for further development so as to reach a realistic result.

Chapter 5

Results

For the purposes of my dissertation project I have tried to bring a part of Nicosia to life. I had been given the generated models of the buildings of Taht el Kale mahalla (neighborhood) that have been produced using architectural rules and the data from the land registry[1]. In the following screenshot you can see the part of Nicosia that I had to work with:

5.1 Map Graph

Figure 5-35: Taht el Kale mahalla (neighborhood). House models were created using architectural rules and land registry data

I have created a graph manually according to the map I had been given, on to which I made adjustments in order to make the houses fit in the corresponding land borders. An input file was created in order to create the graph with its nodes and edges. To do so, I have manually added spheres in every road junction or wherever a person could choose more than one paths to follow. The 3D space positions of the sphere nodes have been written in the file using *writeJunsPos.cs* which is attached to *'Junction'* in the format described in section 4.1 Map Graph. To create the graph edges I had to write manually the connections of the junction nodes. Graph nodes are shown below as red small spheres, whereas edges are shown as red lines connecting the spheres. Besides using as input the mapJunction.txt file that was produced to create the map, new nodes and their edges can be added to the graph by attaching the addNewNode_call script on the Unity Game Object that we want to add as a new graph node. House doors and shops utilize addNewNode_call functionality in order to add all 20 doors and the kiosks to the graph as new nodes.

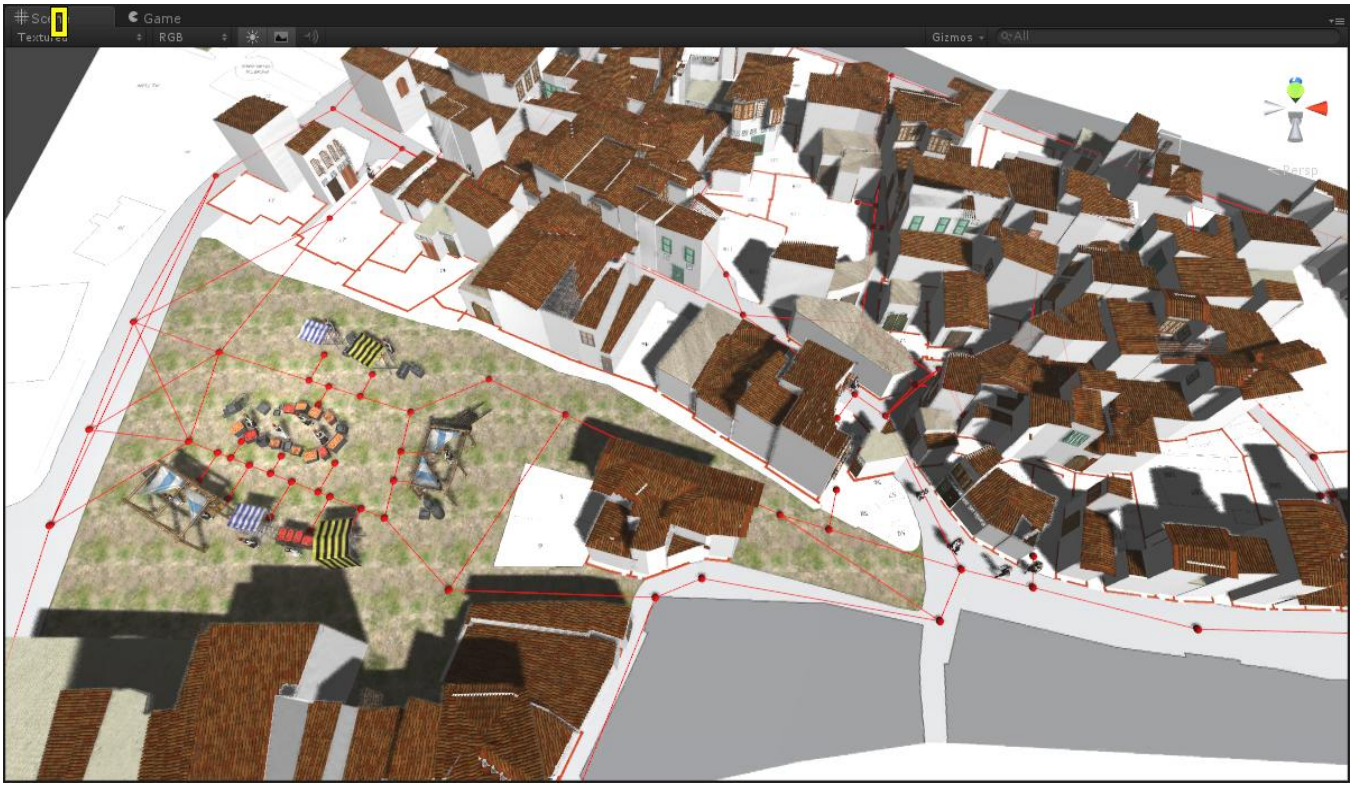

Figure 5-36: Graph produced is denoted in red. Small spheres show the position of nodes, while lines show the connections between the various junction nodes.

5.2 High Level Behavior and Steering

There are two types of behavior for our human models. There are the wandering people and the sales persons. In my implementation there are about 25 wandering persons and 13 salesmen. Wandering people have the DancerScript.cs attached, which is the main controller

of each person, as well as the CypriotAnim animator controller, which works with DancerScript in order to assign the appropriate animations to the character. These people can also have waveTrigger attached, in order to wave at other people when they pass by and addNewTask which enables them to dynamically stop whatever they were doing, add a new task to their list, execute it and afterwards continue from where it was interrupted.

Wandering people are starting their journey from a certain junction point or from a house door and then begin to fulfill the goals written in their taskfile one by one. They are walking through neighborhood's picturesque roads, they may go to the market, negotiate with the salesman and at finally buy something. They are able to talk with each other when they get close enough, dance and of course they can stay idle. Because of the Mecanim animation system transitions between different animations are quite smooth. A person goes from walking to dancing, to buying something to staying idle for a while in a natural realistic way.

In the case that a citizen walks to a node or is standing in idle mode, and the path sets an order for the next destination one being at the opposite direction, the rotation at it, is a bit rough and it is subject to improvement. An appropriate animation in which the human model turns around, could be handy in this case, but unfortunately such animation was not available.

Figure 5-37: Some human models created for this project

When two or more people with the addNewTask script attached, get close enough for their colliders to intersect, they stop whatever task they are executing and they do the task defined in the Inspector. If that task is dance or talk, people turn to look at each other, to do those tasks and hence, when they will continue with the re-entered interrupted task, the rotation is still not a smooth one, for the reasons set above.

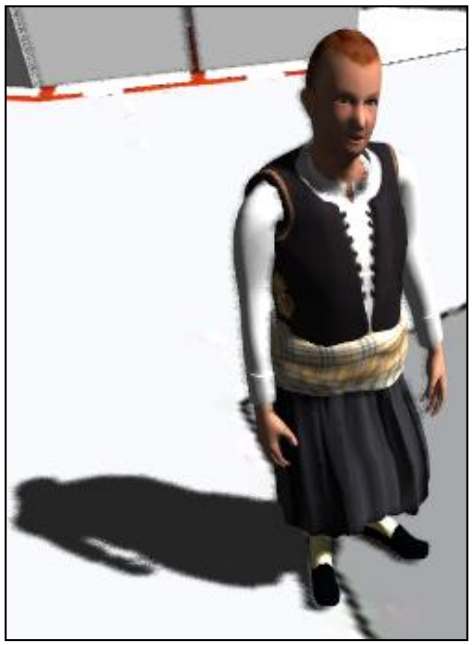

Figure 5-38: A Cypriot human model

There are a few sales people who have salesPersonTrigger attached and salesman animator controller. These people are always in the same place, they are standing by their kiosk or table at the market. Consequently they neither need the walking animation nor the dancing one, for obvious reasons. Salesmen have idle states and the negotiate sub-state animation, the latter, consisting of talk, listen and grab animations in order to negotiate with the buyer and then hand out the goods to them. Also when a move task is followed by another move task, steering is not effective in these cases, as they are two different tasks and have two separate paths.

Anywhere people go they should be able to turn effortlessly so that the movement would look realistic and normal. With the method used in section 4.5.1 they actually rotate towards their next destination node in a pretty good way, which looks natural and realistic. However, in some cases, mostly in the area of the marketplace, where distances between nodes are quite short, people as they are turning unto their target position, sales person forces them to look at them and consequently a jerky movement may be noticed. This is not evident at all times, it depends on the angle of the last two edges of the path and how long these edges are.

The reason for that is that characters may reach their destination before they have fully rotated towards it yet.

5.3 Marketplace

Having an open space, without any buildings, I decided to create a marketplace there, so that our human models would be able to go there for shopping. I have used models from the package Medieval Marketplace[17] which I have downloaded from Unity's Asset Store. The package contained some buildings but these buildings could not be used in my project as

Figure 5-41: Old man sitting in front of his shop[24]

Figure 5-41: The old bazaar in Ermou Street in early 20th century [29]

they do not fit the 19th era and did not look like the Cypriot traditional houses. For that reason I have only utilized the models available that could fit in the marketplace. I have created a market scene with a lot of crates full of potatoes, cherries and apples and a variety of booths as kiosks, where salespeople could sell bread, meat and fish. I have added some additional props for decorating the scene and making it more realistic. Such props were various sizes of barrels and crates, as well as a wheel, a barrow and a bull cart.

Figure 5-39: Screenshot of the marketplace. Graph nodes and edges showing in red

Sales people have a simple job to do. They are standing in idle mode until a buyer (his name starts always with “p_”) gets into their sphere collider, meaning that they have got really close. Automatically the salesman starts talking, touting his merchandise. Buyers walk to the shop and their next task is to negotiate with the seller and then buy something. The negotiate state consists of two sub-modes: first a buyer starts to talk for half the amount of negotiate task duration and for the second half time listens to the salesperson. Hence, the seller will start listening when buyer is in talking mode and will start to talk, when buyer stops talking and starts to listen to them. When buyer stretches their hand to “take” the product bought, so does the sales person and therefore we can observe a realistic shopping transaction.

Figure 5-42: Salesman hands out goods to buyer

Chapter 6

Future Work

6.1 Improvements

The dissertation project I was assigned is part of a broader project. Besides my contribution as part of my dissertation, there are several improvements that could make the project even more realistic and consequently the result would be a more genuine and vivid representation of life in 19th century Nicosia, abiding with the concept of the project.

The overall result is successful and displays a quite natural and realistic movement of life in Nicosia of the 19th century. I need to report two major drawbacks currently in this project which are subject of improvement. Firstly, collision avoidance with buildings. At times it is recorded that agents on a turn attempt may collide with a wall of a building, especially when being in narrow streets. Secondly, the rotation to an opposite direction is not smoothly succeeded. To my consideration an animation where the character turns around naturally could be an appropriate animation to set the image in a more natural movement.

To enrich the whole project several different human models could be added, for instance women, children people of various religions and ethnicities that used to live in Nicosia at that era.

Moreover, using Motion Capture, we could create some additional animations, thus enabling the agents to engage into more complex actions, such as eat something, pay, walk around holding a bag or another prop or check the products. Traditional Cypriot dances could substitute the current mainstream modern dance animation. These animations could clearly be

associated with the job of each person. The data provided by the land registry gives plenty information over the professions of the owner of each house. Therefore we could display an ordinary day of a Cypriot agent starting from the door of his house, to his profession, and observe him eat and sleep.

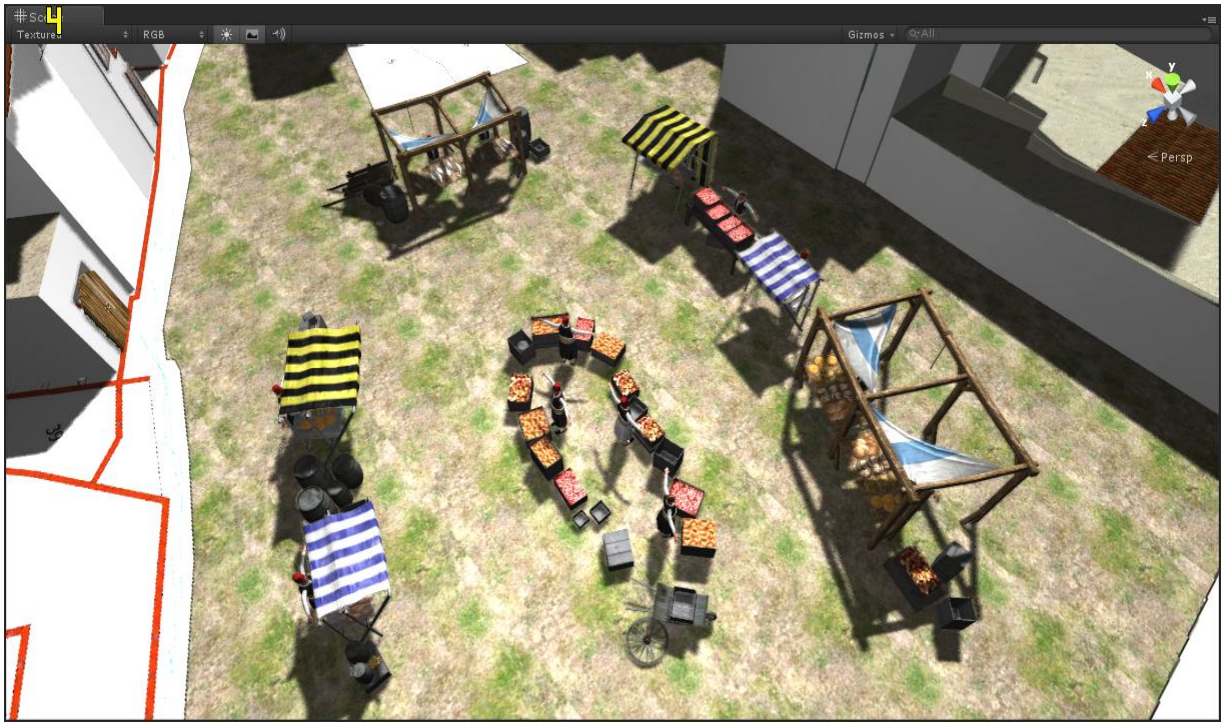

Figure 6-43: A top view of the market place scene

In addition, likewise the commercial game The SIMS[11], we could be able to create a variety of different behaviors triggered by human needs. A person at a time could be thirsty, hungry or sleepy, thus they could do the appropriate actions to satisfy those needs. Appropriate animation tasks like ‘go to well’, ‘eat’ or go to their house to sleep are only some examples of such actions. Furthermore, interaction between human models is another candidate improvement. It would be nice to see characters touch each other, hug, handshake and take some “actual” products in bags or in their hands. It would be interesting to watch people walking or working together, such as lifting

My project is implemented for just a small neighborhood in Nicosia. This could be expanded to all the city are within the walls, as long as there is available information from the land registry about houses, their owners and a digital map for the whole area. Props of the era could be added as well in order to create the physical environment of the city. For instance palm trees and animated animals would give a better sense of what was life in Nicosia in the 19th century.

Hopefully this project will be improved and expanded and as a result a very interesting both educational and entertainment tool, will be created.

Bibliography

[1] C. Panayiotis, I. Hesperia, C. Apostolou, and C. Yiorgos, "Reconstruction of Everyday Life in 19th Century Nicosia," *Progress in Cultural …*, pp. 568–577, 2012.

[2] R. J. Méndez, "UnitySteer - Steering components for Unity." [Online]. Available: http://arges-systems.com/blog/2009/07/08/unitysteer-steering-components-for-unity/.

[3] M. Dikaiakou, A. Efthymiou, and Y. Chrysanthou, "Modelling the Walled City of Nicosia," 2003.

[4] J. Milazzo, N. Rouphail, J. Hummer, and D. Allen, "Effect of Pedestrians on Capacity of Signalized Intersections," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1646, no. -1, pp. 37–46, Jan. 1998.

[5] S. P. Hoogendoorn and P. H. L. Bovy, "Pedestrian route-choice and activity scheduling theory and models," *Transportation Research Part B: Methodological*, vol. 38, no. 2, pp. 169–190, Feb. 2004.

[6] L. F. Henderson, "On the fluid mechanics of human crowd motion," *Transportation Research*, vol. 8, no. 6, pp. 509–515, Dec. 1974.

[7] G. G. Lovas, "Modeling and simulation of pedestrian traffic flow," *Transportation Research Part B: Methodological*, vol. 28, no. 6, pp. 429–443, Dec. 1994.

[8] N. Pelechano, J. M. Allbeck, and N. I. Badler, *Virtual Crowds: Methods, Simulation, and Control*, vol. 3, no. 1. Morgan & Claypool Publishers, 2008, pp. 1–176.

[9] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, pp. 4282–4286, 1995.

[10] W. Shao and D. Terzopoulos, "Autonomous pedestrians," *Graphical Models*, vol. 69, no. 5–6, pp. 246–274, Sep. 2007.

[11] "The SIMS 3." .

[12] E. W. Weisstein, "Cellular Automaton," *MathWorld*.

[13] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 25–34, Aug. 1987.

[14] J. Maïm, S. Haegler, and B. Yersin, "Populating ancient pompeii with crowds of virtual romans," *… Symposium on Virtual …*, vol. 0, no. 0, 2007.

[15] G. Ryder, P. Flack, and A. M. Day, "A Framework for Real-Time Virtual Crowds in Cultural Heritage Environments," 2005.

[16] “Unity.” [Online]. Available: http://unity3d.com/unity/.

[17] “Medieval Marketplace.” [Online]. Available: https://www.assetstore.unity3d.com/#/content/1812.

[18] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” in *Numerische Mathematlk 1*, 1959, pp. 269–271.

[19] Wikipedia, “System of linear equations.” [Online]. Available: http://en.wikipedia.org/wiki/System_of_linear_equations.

[20] “A* Pathfinding Project.” [Online]. Available: http://arongranberg.com/astar/.

[21] “A* Pathfinding Project - Grid Graph.” [Online]. Available: http://www.arongranberg.com/astar/docs/graph_types.php#grid.

[22] “A* Pathfinding Project - Point Graph.” [Online]. Available: http://www.arongranberg.com/astar/docs/graph_types.php#point.

[23] “A* Pathfinding Project - NavMesh.” [Online]. Available: http://www.arongranberg.com/astar/docs/graph_types.php#navmesh.

[24] “The Leventis Municipal museum of Nicosia.” Nicosia.

[25] “Airport Security - SMART Solutions Network.” [Online]. Available: http://www.smart-solutions-network.com/photo/cardiff-city-centre-2.

[26] H. Yeh, S. Curtis, S. Patil, J. Van Den Berg, D. Manocha, and M. Lin, “Composite Agents,” 2008.

[27] Anthony Capobianchi, “UNITY 4 – LOOKING FORWARD.” [Online]. Available: http://blog.infrared5.com/2012/09/unity-4-looking-forward/.

[28] “Game/AI: Fixing Pathfinding.” [Online]. Available: http://www.ai-blog.net/archives/000152.html.

[29] Z. M. Rena Hoplarou, “Nicosia is calling … Chrysaliniotissa Neighborhood.” 2012.