Final Game Design Document

Section 1: Execute Summary

1.1 Focus of the game

The game is a real-time strategy (similar to SimCity), where the player is a network designer and administrator of a large telecommunications firm. The player's goal is to provide efficient digital communication service to customers in terms of customer satisfaction and profit by building a cost effective and reliable network.

1.2 Educational purpose

The game aims to teach fundamental techniques involved in the design of digital communication networks. It does so by introducing elements normally found in those networks (such as cables, and routers), allowing player to experiment with them. By analyzing customer's needs, and designing the network to meet them, player can develop the critical thinking skills necessary to succeed in network design and support tasks.
The game is not meant to replace a computer networking course, but it is meant to provide an interactive learning experience that allows students to apply their knowledge in a secure, challenging, and fun environment.

1.3 Game mechanic (Gameplay)

1.3.1 Gameplay style

The game is a real-time strategy (RTS). Game starts with a screen of an empty terrain. As time progresses, player will see customers (represented by small houses or businesses) appearing randomly on the terrain. Each customer specifies three requirements that needs to be met: peers (a set of customers that this particular customer would like to communicate with), the minimum bandwidth, the monthly fee the customer is willing to pay.

1.3.2 Game elements

The player's job is to lay digital cables on the map and place routers that interconnect them by choose from a set of available tools. The player's primary interaction method is select-n-drop, where player selects one of available tools, and uses the mouse to place a cable or router on the terrain, similar to SimCity. The available tools are divided into two categories: *cables* and *routers*. There are three types of available cables that differ in their maximum bandwidth and cost per mile:

1) 1 Mbps
2) 10 Mbps
3) 100 Mbps

Similarly, there are three types of available routers that differ in the number of interfaces, maximum

bandwidth the router can handle, and cost per unit:

1) 4 Interfaces, 16 Mbps
2) 6 Interfaces, 64 Mbps
3) 8 Interfaces, 265 Mbps

1.3.3 Rules

At the beginning, the game will specify the number of customers that the player has to connect before winning the game. This number is based on the difficulty level. Player starts with a balance of a certain initial capital, which is also based on the difficulty level. The player has to design his/her network so that to ensure the existence of a connection (possibly passing through intermediate routers) between each customer and his set of peers customers. At that time, the player may negotiate the monthly fee with the customer. If the negotiation succeeds, the customer becomes subscribed (contracted) to the service and will start paying the fee.

Once a customer is contracted, traffic will start flowing from his cable, and through the network to his peers. The player has to take care to contract all the peers of the customers as soon as possible, otherwise, the customer will not be very happy. Every element of the network (both cables and routers) will have a *utilization indicator*, that shows the current bandwidth consumption on that element. The utilization indicator of an element will be shown as a bar that appears above the selected element. If the utilization exceeds the maximum bandwidth of the element, packets will be dropped, causing customers to suffer.

Each customer has an associated satisfaction rate that changes with time. This rate is calculated as a percentage of the met requirements, that is, how much bandwidth the player is supplying to customer vs. his minimum requirement, and to how many peers the customer can communicate seamlessly with. The overall customer satisfaction is calculated as an average of the satisfaction rates of individual customers. If this overall percentage drops below a certain level (20 %), the player immediately looses the game. The player also looses the game if the balance drops below zero

1.3.4 Goal

The goal of the game is to connect all customers that appear, and attain a satisfaction rate of more than 50%. When the player manages to contract all customers attaining the minimum overall satisfaction rate, they player wins.. However, the balance can not go below a certain negative amount. If this happens, the player loses.

1.3.5 Challenges

As more and more customers are added, the network gets more congested more, and the player needs to add more cable, and possibly replace old routers with new ones of higher maximum bandwidth and/or interfaces.

Section 2: User interface and controls

Our game interface has been designed to facilitate ease of navigation on the screen and allow player to see all aspects of the game within 2 mouse clicks or less. Our pre-game menus are easily navigated by use of keyboard arrows and Enter key. In the game, the user uses mouse to select tools and objects on the screen and 3d map, and uses keyboard control to rotate or move around the map. Zoom in/out functions can be performed by both the keyboard and a mouse wheel.

In SimNetwork we've paid a lot of attention to widgets. Most of the game is widget based. The use of widgets allows player to see which customer's peers are not connected yet, player can check status of cable or router at any time. The negotiation process is widget based, in which we make it easy for the player to contract a new customer (in negotiation process, player uses both keyboard and mouse to complete the negotiation). We've use widgets also to implement an in-game help system. By pressing H key, player can see the control screen that helps him or her play the game.

The third element that we've thought was important in our game is the messaging system. Player should be informed at any point in the game what is going on. We use a message field right below the toolbar to display relevant messages to the player (complemented with sound, so increase attention to the message), we use timer field to show player how much time has passed, and we show balance, customer satisfaction, and customers left fields to allow the player to see how well is he or she doing in the game at any particular moment.

The toolbar has been strategically placed on the top of the screen, where is does not obstruct the view of the current game screen. The original design placed toolbar on the bottom of the screen, but after user testing we found that it can interfere with the scene and more scene scrolling is required than when it's placed on top of the screen.

Section 3: Implementation

3.1 Scene programming

3.1.1 Mouse picking

Mouse picking was implemented with Electro's built-in collision detection system. To detect the 3D entity currently under the cursor, a virtual ray the passes from the 3D camera to the coordinates of the mouse cursor is created. Reported collisions are first queued into a special vector. The queued collisions are then analyzed be a separate the do_timer() thread which performs the appropriate action such as drawing a cable or dropping a router.

3.1.2 Collision avoidance

Electro's collision detection system was also used to restrict the player from dropping elements over other existing elements. The mechanism used for collision avoidance is identical to the one used for mouse picking

3.2 Game logic

In designing SimNetwork's game engine, we have tried to follow an object oriented approach that emphasized the decoupling of presentation from the game logic. The motivation of this approach stems from our desire to make the game independent from the underlying visualization engine as much as possible. This serves two purposes

1. The logic subsystem could be regarded as an independent module. Thus, its behavior could be modified without affecting the presentation. This allows us to easily replace the current simulation algorithm with different ones.
2. Should we decide to port the game to another presentation platform (OpenGL, for example), the game logic would not need any modifications, and the porting time would be much less.

3.2.1 Basic building objects

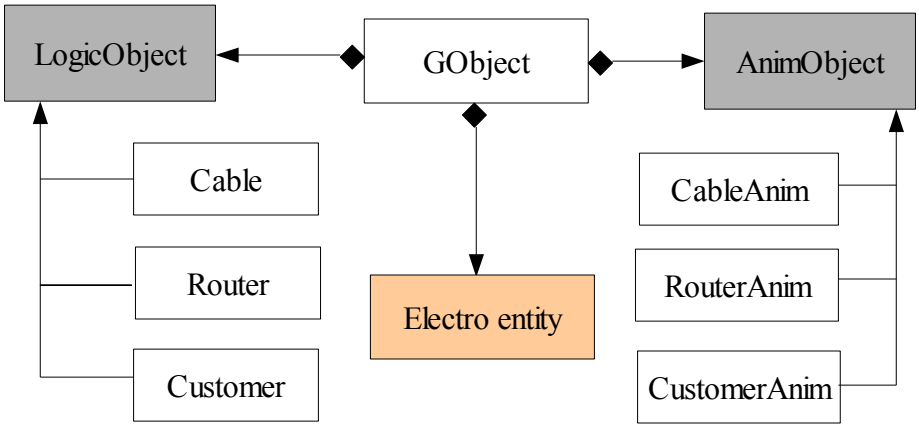


Figure 3.1: SimNetwork's data structure in UML notation

Each element in the game is represented by one GObject (for Game object). The GObject works as a container that maintain 3 references:

- A reference to a LogicObject: this is the object that encapsulates the logical status and behavior of the entity.
- A reference to an Electro entity used to visualize the element
- anim: a reference to an AnimObject: this object handles the animation of the element

LogicObject is an abstract class. Since our game has three different elements (cable, router, customer), we have three classes that inherit from this class:

- Cable: among many things things, this object contains variables that tells the current utilization of the cables, which elements (customers or routers) this cable connects, etc..
- Router
- Customer

All of the these three objects share some common methods inherited from LogicObject, and each of them adds a set of unique methods corresponding to its behavior.

The reader can infer from diagram 5.1 the three different classes that handles the animation for each type of element:

- CableAnim: handles cable animation
- RouterAnim: handles router animation
- CustomerAnim: handles customer animation

3.2.2 Network representation

The player's network is represented by an undirected graph, called the links graph. The nodes of the links graph represent customer or routers, and the edges represent cables. The nodes are undirected

since the traffic can flow in two directions on one cable. As we will see later in this chapter, this fact will contribute significantly to the complexity of the simulation problem. The graph is represented with an adjacency list: each node lists all other neighboring nodes, along with a reference to the cable the connects it to that neighboring edge.

3.2.3 Peer relationship representation

Since the peer relationship is asymmetric, we represent it using a directed graph called the peers graph. This graph is also represented with an adjacency list: each customer lists all other peer customers.

3.2.4 Simulation Engine

The simulation engine is composed of two subsystems:

- Traffic simulation algorithm: determines the loads on the cables and routers given the current requirements of the customers.
- Fuzziness engine: generates random events such as creating a new customer, updating access pattern of an existing customer, etc...

The traffic simulation algorithm as executed on as-needed basis, while the fuzziness engine is executed at regular basis from the game loop. By remembering the results of previous simulation iterations, we have tried to minimize the calls to the traffic simulation algorithm, because as we will see later, a single call incurs a big overhead and it would be almost impossible to execute the simulation algorithm at every game loop iteration without significantly degrading the frame rate.

3.2.4.1 Traffic simulation algorithm

One of the most challenging parts of SimNetwork was the traffic simulation algorithm. The complexity of the underlying algorithmic problem stems from the difficulty of finding an optimum (or close to optimum) instance from a set of exponentially large number of “feasible” flows

The algorithm takes the links graph and the peers graph, and outputs a “feasible” flow on the links graph. A feasible flow is one the respects the following conditions:

- On every cable, the flow does not exceed the maximum bandwidth of the cable.
- On every router, the incoming flow does not exceeds the router's maximum routing bandwidth.
- Each customer wants to communicate with all of his/her peers with an equal amount of traffic units (Mbps). The term communicate is vague: is it one way, or two way, or both? However, we can simplify this as follows: suppose the current required bandwidth of a customer is N, and the customer has K peers. Then, the customer wants to send N/K Mbps to each peer.

We will first demonstrate the different approaches that we tried, then proceed to explain the solution that we developed. The impatient reader can skip sections 3.2.4.2 and 3.2.4.3, and proceed directly to section 3.2.4.4 for the final solution.

3.2.4.2 Is this an NP-Complete problem?

We tried to reduce our simulation problem to similar problems described in the literature.

At first, our algorithm seemed as a generic flow network problem [Kleinberg] with the objective of finding the maximum flow. However, our simulation algorithm requires a different type of traffic for each customer, and the generic flow problem assumes a single type of "material" flowing in the network.

An extension to the flow problem that satisfies our requirement of different types of traffic is the multi-commodity circulation problem [CLRS]. The only known solution to this problem is linear programming. However, there is still a fundamental difference between our problem and the multi-commodity circulation problem: we are working on the undirected links graph and the latter is working on directed graphs. One might first suggest that for each undirected edge (cable) in the links graph, we add two directed edges with the same capacity as the undirected edge. However, by doing this, we would be effectively doubling the bandwidth of each cable. Another suggestion is to have the sum of the capacity of the two directed edges equal to the bandwidth of the cable. But we're faced with a number of choices equal to the bandwidth of the cable. Selecting any of these choices would reduce the space of feasible solutions by at least that same number. Also, we can not represent the notion maximum routing capacity of a router with an analogous notion in the multi-commodity problem. And to make matters worse, the multi-commodity problem mandates that each source node be mapped to one sink node. However, in our setting, a customer needs to send traffic to several other customers (peers), not just one. Also, a customer might act as a source and a sink at the same time.

3.2.4.3 Reducing the simulation algorithm to the multi-commodity circulation problem

Despite the differences outlined above, we tried to reduce our simulation problem to the multi-commodity circulation problem. Here's how:

1. For each edge in the peers graph $e_i(c, p) \in G_p$, add a source and sink nodes s_i, d_i respectively, and add the following two directed edges $l(s_i, c), l(p, t_i)$ to the links graph with an infinite capacity. This effectively encodes the notion of overlapping sources and sinks in the multi-commodity problem.
2. For each cable: add the two directed edges with a capacity equal to half the bandwidth of the cable. By doing this, we have sacrificed a significant portion of the feasible flow space solution. But, we had to make the trade-off.
3. Remove the constraint on the routing capacity of the router: this was a painful trade-off, since the routing capacity was a fundamental rule in the game play.
4. Generate a linear program with a maximum optimization goal.

This worked very well in theory. However, the number of linear conditions in the resulting linear program was $O(NM)$ where *N* is the number of customers, and *M* is the number of cables. This meant that the simplex algorithm is a polynomial function of another polynomial function of the input. Taking into consideration that simplex algorithm is polynomial in best cases and exponential in worst cases, there's no chance in using it as a real-time simulation algorithm.

3.2.4.4 Final solution

The solution we have devised does not sacrifice any of the game rules. However, it does not guarantee an optimal solution. This turned out to be a very reasonable trade-off, since we're only interested in a solution that would make sense in the game, not an optimal one. The following is a high level description of our algorithm.

SimulateTraffic(LinksGraph, PeersGraph)

```
For each contracted customer i do
        Gi = Construct a generic flow network graph using a similar process to
        the one outlined in section 4.2.1.2.
        Find the maximum-flow on g using the Ford-Fulkerson algorithm
        Store graph Gi
End For

// At this point, we have obtained one maximum flow for each customer
// individually, but these flows are not valid when combined together, since
// it is very likely that at some point, they'll exceed the capacity of a cable.
TempFlow = CombineFlow(G1, G2, ..., Gn)

// The recap process searches for cables/router which have a utilization exceeding
// 100%. Such elements are marked as “high contention” points. Each high          // contention
point is subjected to a “push back process” that redistributes the
// bandwidth on the contending parties, decreasing the utilization to %100. The
// process is repeated until we have eliminated all contention points. This proces
// guarantees to eliminate all contention points since it decreases the flow at every
// iteration.
FinalFlow = RecapFlow(TempFlow)

Return TempFlow
End
```

The time complexity of the Construction and CombineFlow process is estimated at $O(nm^2)$, where n is the number of customers, m is the number or routers. The complexity of RecapFlow is estimated at $O(m^2)$. The whole traffic simulation algorithm is run on as needed-basis. That is, when the structure of the network changes, or when the access patterns are updated.

One final interesting aspect to note is the the effect of the CombineFlow and RecapFlow() with respect to the attained flow. Figure 3.2 characterizes that effect. As we can see, the flow is increased to an unfeasible value as the CombineFlow is progressing. However, when the RecapFlow iterations start, the flow guaranteed to returned to a feasible value.

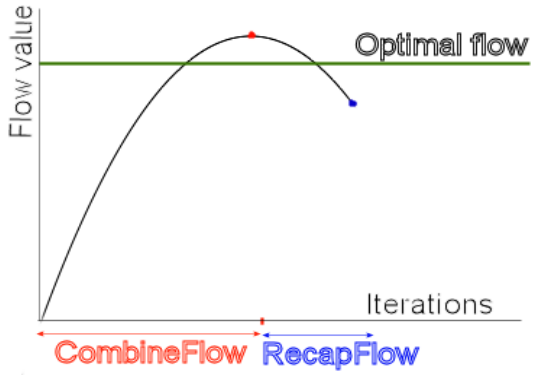


Figure 3.2: Effect of progression of simulation algorithm on attained flow

Acknowledgment

Some of the ideas for the traffic simulation algorithms were suggested by computer science professor Tanya Berger-Wolf at the University of Illinois at Chicago.

3.2.4.5 Fuzziness Engine

The fuzziness engine is what makes the game play challenging and exciting. It is responsible for generating random events, such as creating a new customer and updating the access patterns of an existing customer, or adding a new peer to an existing customer. Since the fuzziness engine should be executed at regular intervals, we had to adopted decisions that simplified both the implementation and time complexity. The first decision was to have the fuzziness engine totally base its events on a pseudo-random number generator. This means that the engine will not try to analyze the current state of the game or assess how well the player is going before taking its decisions. This is probably not the best approach, as a game would be more exciting if it increased the difficulty when the player is doing well, and decrease it when he/she is doing bad, but this would serve our purpose for a demo game.

We will describe how the fuzziness engine creates a customer and assigns it random access patterns and requirement fee.

The possible access patterns and types requested monthly fee for different types of customers are specified in a trapezoidal fuzzy distribution function.

Figure 3.3 shows the generic form of the trapezoidal fuzzy distribution function [Murata]:

> "The fuzzy distribution function $\pi(\tau)$ gives the numerical estimate of the possibility that some event will happen at time π . When $\pi(\tau)=1$, it is completely possible that some event will happen at time π . When $\pi(\tau)=0.5$, there is '50-50' chance that it will arrive at time π "

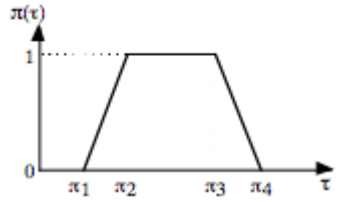


Figure 3.3: The trapezoidal fuzzy distribution function

We use the special case of this distribution where $\pi_1 = \pi_2$ and $\pi_3 = \pi_4$, and we assume only discrete values for τ. Let us take an example: the bandwidth of the customer of type "offices" is described with $\pi_1 = \pi_2 = 1$ and $\pi_3 = \pi_4 = 21$, that is, bandwidth requirement of this type of customers can range anywhere from 1 to 21. As we noted earlier, we allow only allow discrete values. We also specify a "density" factory that specify the distance between two discrete values.

References

[Kleinberg] Jon Kleinberg, and Éva Tardos, "Algorithm Design", Addison Wisely (2005)

[CLRS] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, "Introduction to Algorithms, second edition", MIT Press (2001)

[Murata] T. Murata, "Temporal Uncertainty and Fuzzy-Timing High-Level Petri Nets", Application and Theory of Petri Nets 1996, Springer-Verlag

Section 4: Graphics

4.1 Models

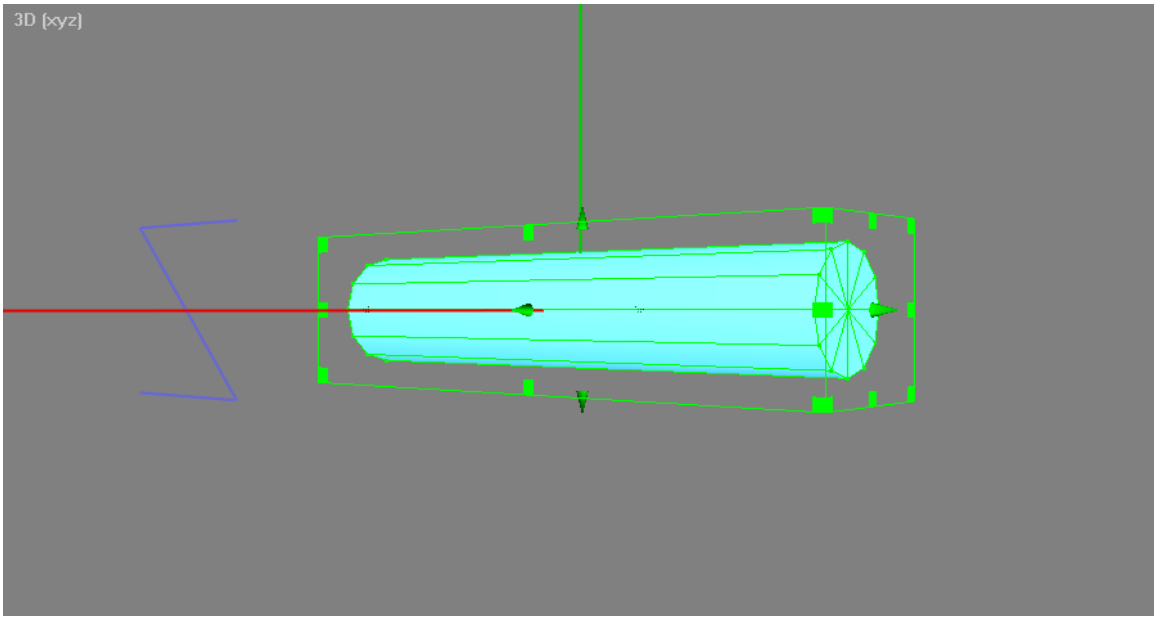
All the 3D models have been constructed in AC3D. The game comprises of broadly the following models:

- Cables
- Routers
- Customers

4.1.1 Cables

Three types of cables have been considered in the game.

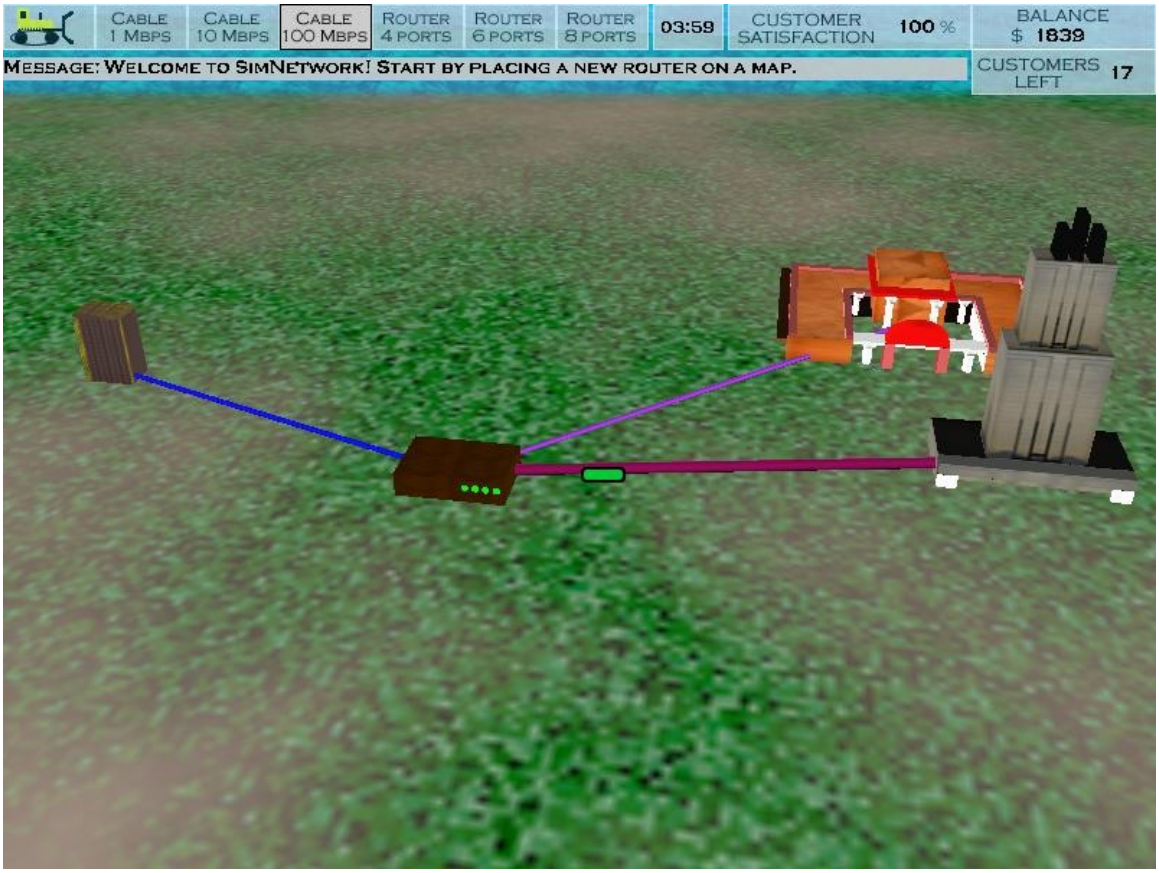
4) Cable with data transfer capacity of 1Mbps.
5) Cable with data transfer capacity of 10Mbps.
6) Cable with data transfer capacity of 100Mbps.



Screenshot 1: Cable

With increasing amounts of capacity the both the color and the size of the cable varies. While connecting the cable between customers, it appears in sky blue color (Screenshot1). After the connection is established each type of cable has a different color.

- 1Mbps Cable: appear in blue color after the connection has been established.
- 10Mbps Cable: appear in lavender color after the connection has been established.
- 100Mbps Cable: appear in purple color after the connection has been established.



Screenshot 2: Cables and Router

4.1.2 Routers

Three types of routers have been considered in the game.

4) Router with 4 ports.
5) Router with 6 ports.
6) Router with 8 ports.

All the three types of routers are quite similar. In the above screenshot a four-port router can been seen as a brown box with green lights.

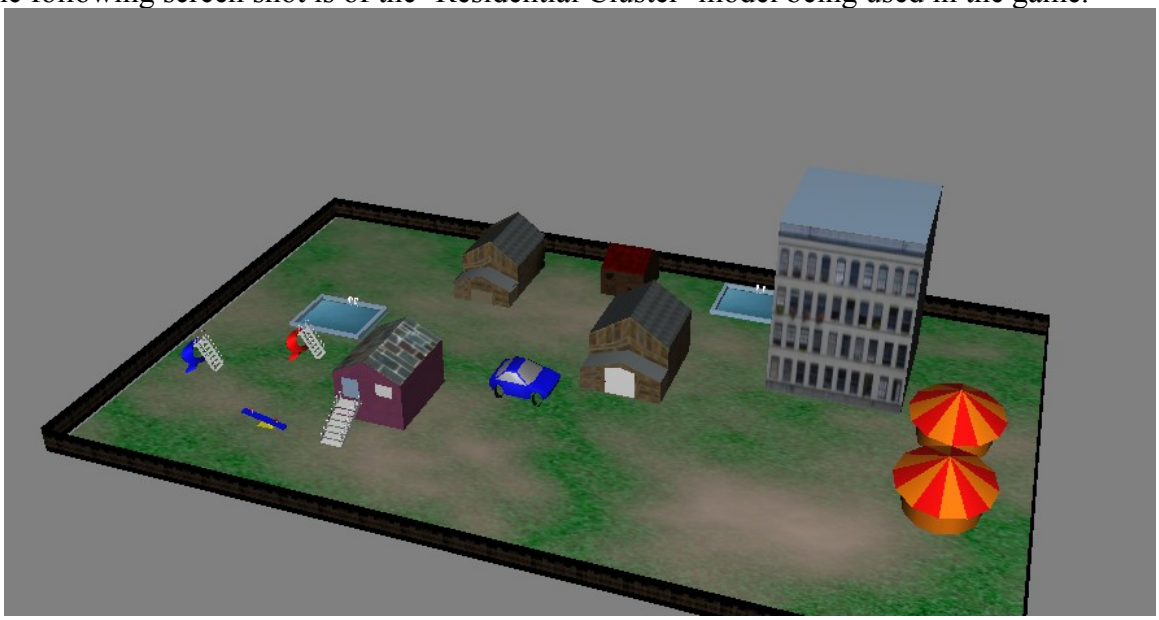
4.1.3 Customers

The main categories of customers have been taken into consideration are:

- Residential
- Educational
- Industrial/Commercial

4.1.3.1 Residential Neighborhood

The following screen shot is of the 'Residential Cluster' model being used in the game.

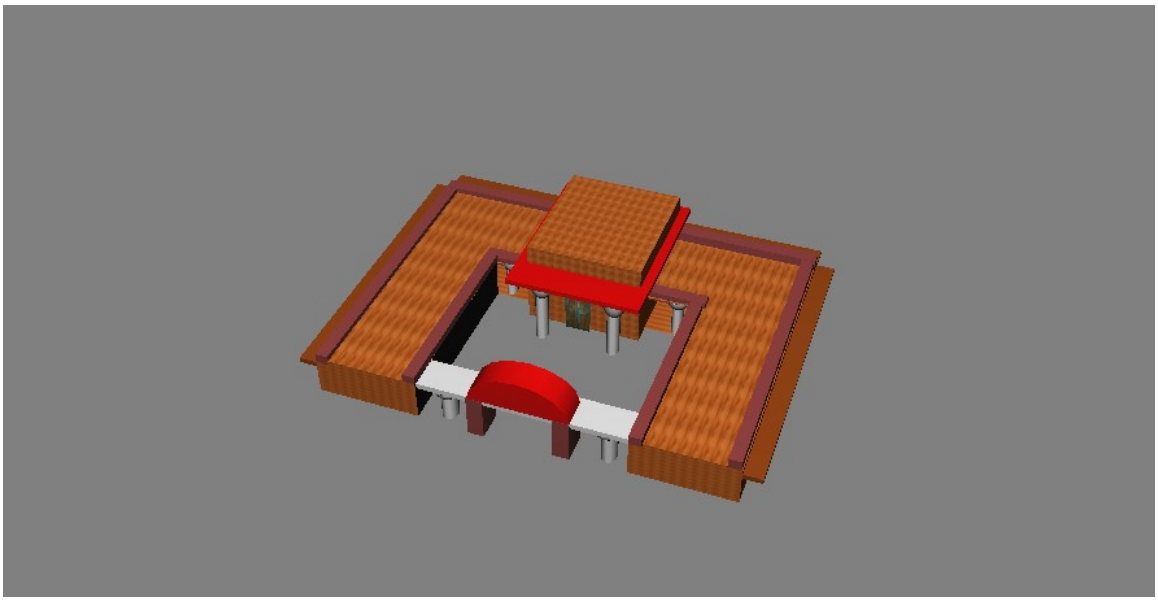


Screenshot 3: Residential Cluster

4.1.3.2 University

The following screenshots are of the university models being used in the game.

Screenshot 4: University1

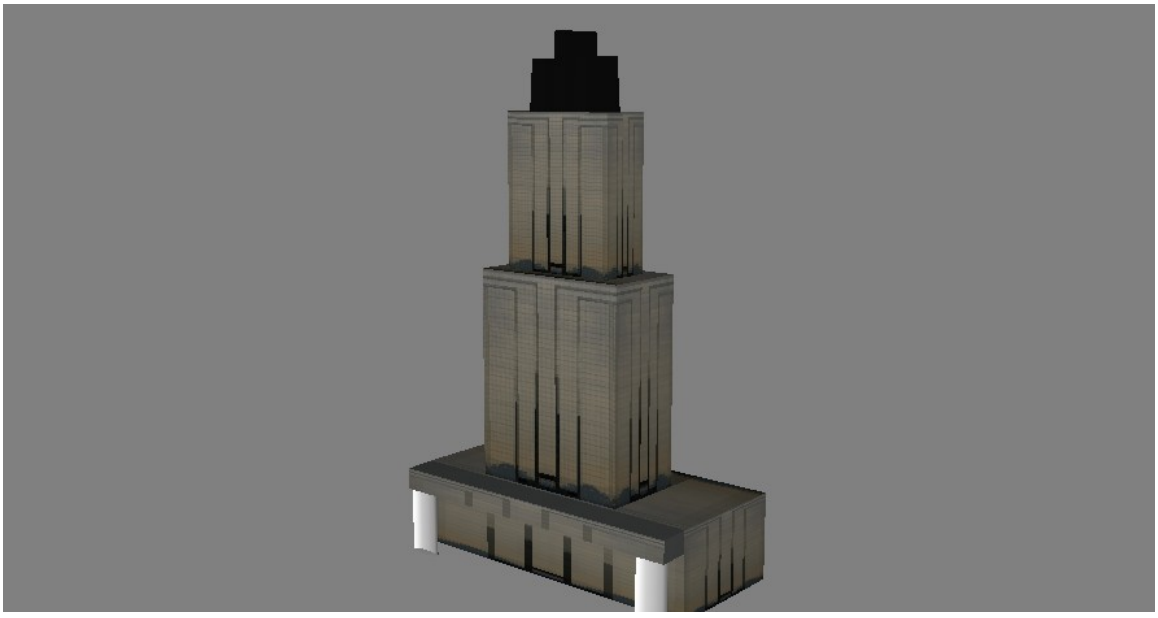


Screenshot 5: University2

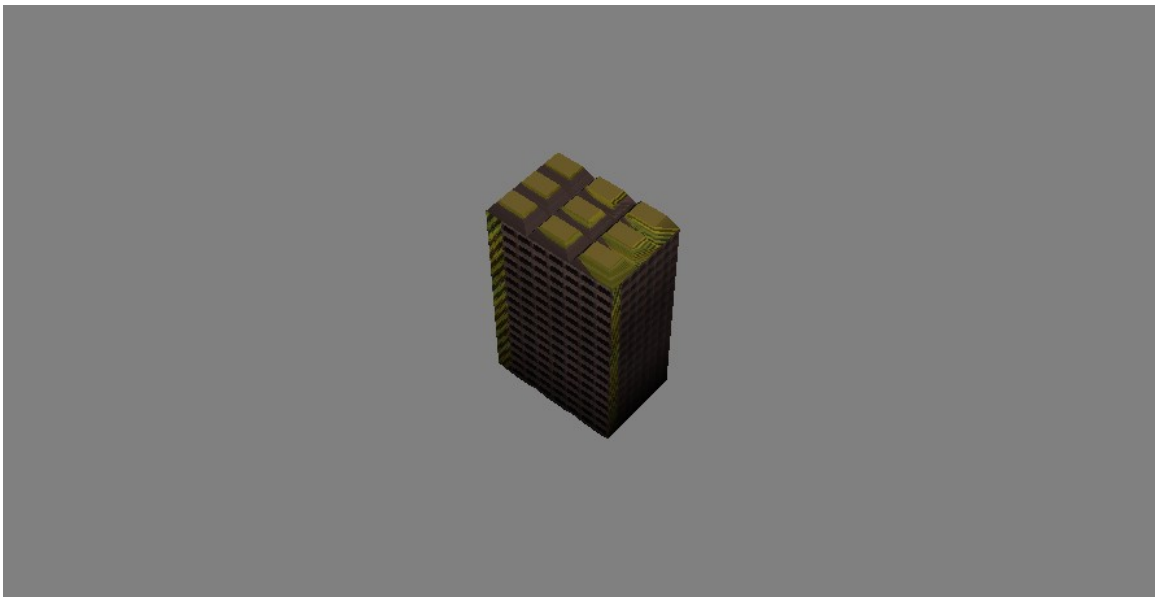
The model in the Screenshot 3 has been inspired from a real life University.

4.1.3.3 Corporate

The following Screenshots show the Corporate office models being used in the game.

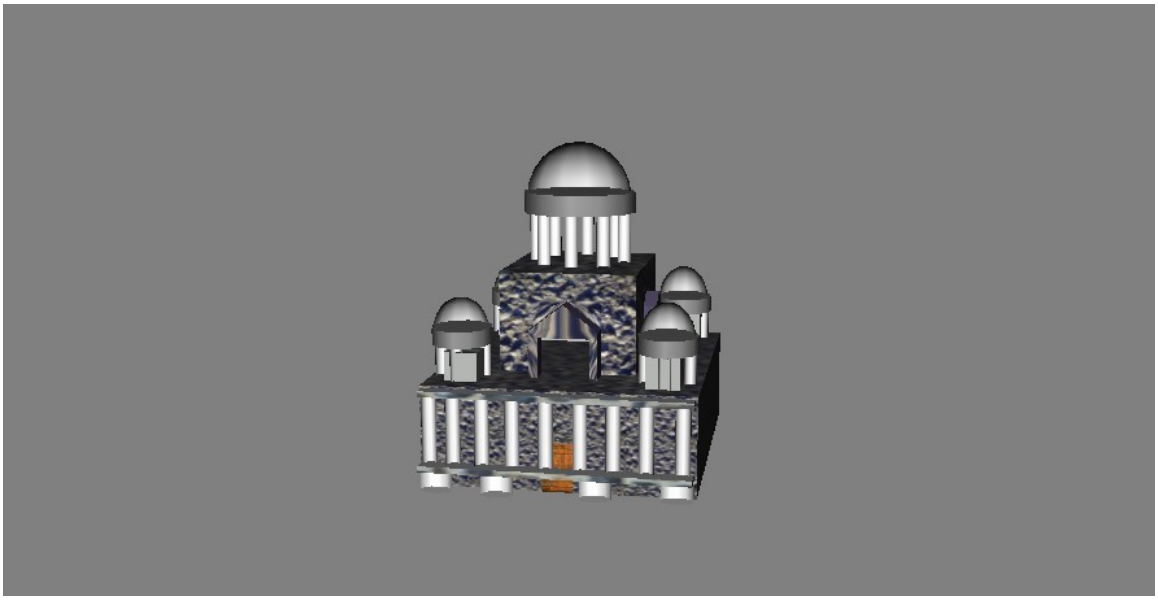


Screenshot 6: Corporate Office



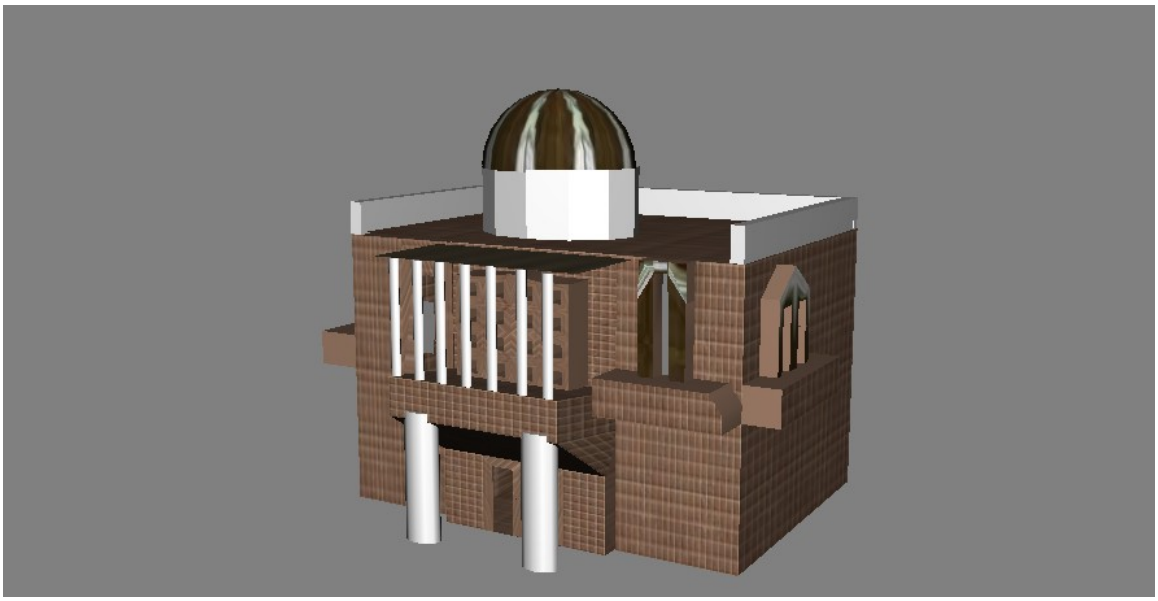
Screenshot 7: Office

4.1.3.4 Library



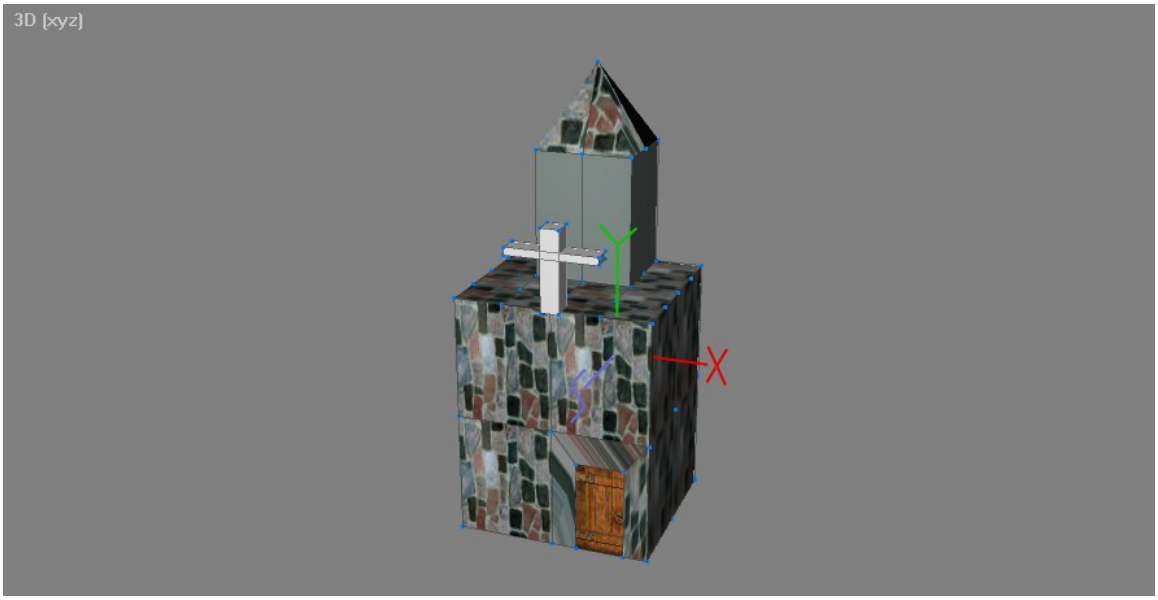
Screenshot 8: Library

4.1.3.5 Community Hall



Screenshot 9: Community Hall

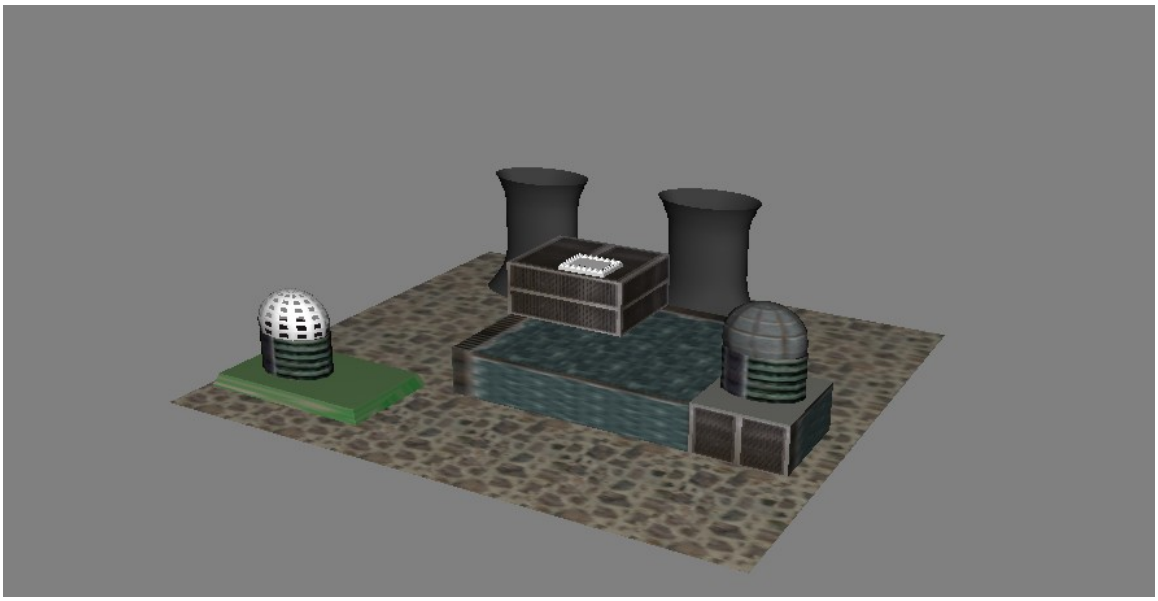
4.1.3.6 Church



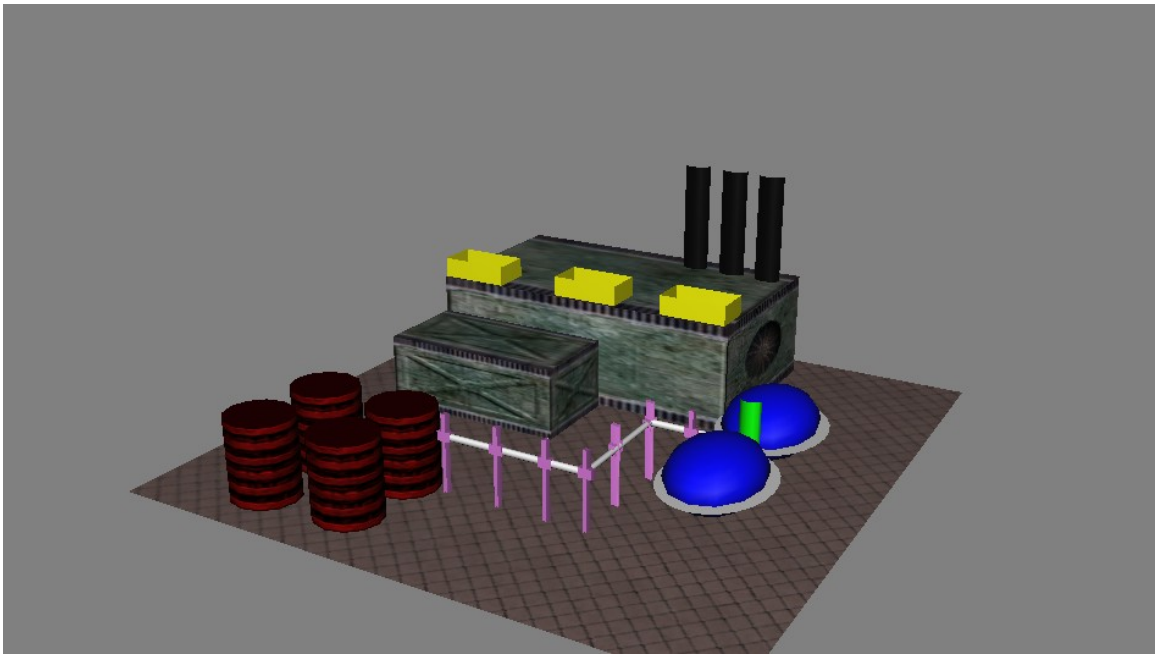
Screenshot10: Church

4.1.3.7 Industries/Factories

The models in the Screenshots 11 and 12 have been inspired from the models from Simcity 2000.

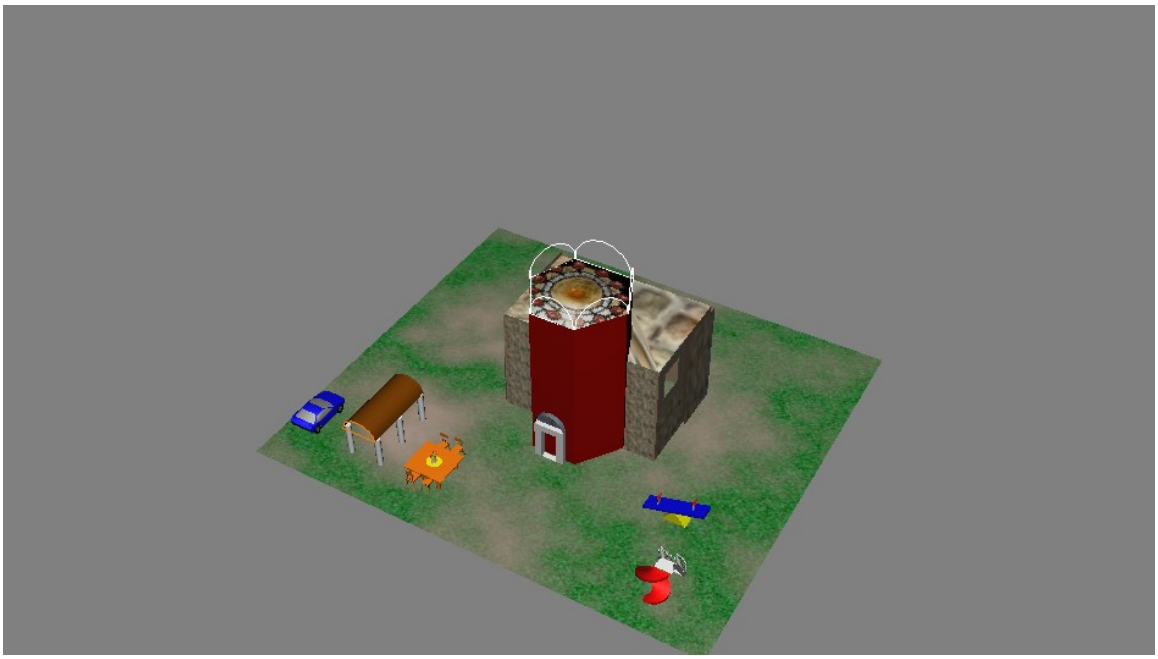


Screenshot11: Factory1



Screenshot12: Factory2

4.1.3.8 Research Lab



Screenshot13: Research laboratory

Section 5: Sound effects and music

5.1 Sound effects

Sound effects used in a game include effect generated by Foley artist as well as effect that were obtained under royalty free licenses.

Sound effects include clicks on the toolbars, menu selection, demolish tool, selecting and deselecting objects on the screen, purchasing a cable or router, message indicator, and new customer indicator. In addition, a sound effect of clapping is played if player wins the game, and a sound effect of "woohoo" is played when player looses the game. Motivation for the latter is to make loosing the game fun – if you feel good about loosing, you will probably play the game again.

5.2 Music

Music was obtained from a recording of a live concert as part of the Syrian Jazz Festival in Damascus, in 2005. The permission to use the music was obtained from the sound engineer, Mr. Kinan Admeh. The music was specifically selected to create a mood of being immersed in jazz environment. Jazz environment music works well with SimCity type of games, and we have used this fact. There is different tune for the menu and for the actual game. The music switches to the right tune depending if the player is playing the game, or browsing through the menu.

Conclusion and Future Development

SimNetwork is the first serious game the focuses totally on teaching strategic skills involved in the design of modern computer networks.

Although the current game features simple rules, the game could be expanded to include new parameters, such as quality of service, latency, or jitter. More advanced and accurate simulation algorithm could be developed to aid researchers. We believe that the game has a potential to become a valuable performance evaluation tool.

One setting in which we envision future versions of SimNetwork being applied is the design and evaluation of distributed real-time multimedia systems. The design of such systems is difficult due to their inherited complexity and the tight requirements. The availability of easy to use, high level performance evaluation tool would undoubtedly make the life of engineers working on these systems much easier.