Believable communication between non player characters

Pedro Roquette Quadros Saldanha

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Carlos António Roque Martinho

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Carlos António Roque Martinho
Member of the Committee: Prof. João Miguel De Sousa de Assis Dias

November 2018

Acknowledgments

I would like to start by thanking my advisor Carlos Martinho, first and foremost, who has always given me great feedback/suggestions when I needed them the most, and for being always available with his time, effort and dedication. This work would not have been possible without him.

To my friends and colleagues, I would like to thank you for supporting this project with game testing and good feedback.Your ideas helped my work take the shape that it did, and I am sure some of those ideas are present in the final work. So, for the honest and valuable support you've freely given me, I thank you deeply.

I would also like to thank my parents and brothers, who always believed in me through all the years I have been studying. They gave me their friendship, encouragement and caring over all these years, always helping me when I needed and keeping me focused on what was important at a time when I felt lost and without motivation.

I would also like to make a special acknowledgment to 'Instituto Superior Técnico', a place that feels more like a second home to me than a place of work and research. A special thanks to all the professors who taught me and influenced me into becoming the person I am today. A big thank you.

Abstract

Motivation for this work comes from the belief that there is some untapped potential is giving non-player characters (NPCs) in video games the ability to share knowledge between themselves. This dissertation proposes a model for how NPCs in video games can communicate between themselves in a more believable way.

This model gives NPCs the ability to perceive events from their surrounding environment, allowing them to change their behavior based on those events. We believe that by controlling how the information generated by the world and the player is shared over time among NPCs, interesting behaviors can emerge that give a richer gameplay experience to the player.

The focus on time as the most important variable for how information is shared in a simulated world, where distance is taken into account, allows for different players do reach different game states, based on where and when did they perform their actions. Not only will different players perceive the world differently from each other but the way in which the agents(NPCs) in the game world perceive the player will change.

In this multi-agent system of NPCs, we make use of decentralized systems to give each individual NPC the ability to make his decisions independently from the other NPCs and from this decentralized behavior emergent gameplay situations can happen.

We detail here the different techniques game designers can use to make a system like this one work, the difficulties they might encounter during development and how they can increase the complexity of NPCs.

Keywords

NPC communication, message propagation, emergent behavior, video game, believable communication

Resumo

A motivação para este trabalho vem da crença de que existe um potencial inexplorado em dar aos personagens não-jogadores (NPCs) em videojogos a capacidade de partilhar conhecimento entre si. Esta dissertação propõe um modelo de como os NPCs em videojogos podem comunicar de uma forma mais credível.

Este modelo dá aos NPCs a capacidade de percepcionar eventos de seu redor, permitindo que eles mudem seu comportamento com base nesses eventos. Acreditamos que controlando como a informação é gerada e transmitida ao longo do tempo entre os NPCs, podem surgir comportamentos interessantes que proporcionam ao jogador uma experiência de jogo mais rica.

O foco no tempo como a variável mais importante para como a informação se propaga no mundo e onde a distância é tida em conta, permite que diferentes jogadores alcancem estados de jogo diferentes, com base em onde e quando eles realizaram suas acções. Não só iram jogadores diferentes ter uma percepção diferente do mundo ao longo do tempo mas tambem a percepcao que os NPCs têm do jogador ira mudar.

Neste sistema multiagente de NPCs, utilizamos sistemas descentralizados para dar a cada NPC individual a capacidade de tomar suas decisões independentemente dos outros NPCs e, a partir deste comportamento descentralizado, abrindo a possibilidade a comportamentos emergentes de jogo.

Nós detalhamos aqui as diferentes técnicas que designers de jogos podem usar para fazer um sistema como este funcionar, as dificuldades que eles podem esperar encontrar durante o desenvolvimento e como podem aumentar a complexidade dos NPCs.

Palavras Chave

Comunicação entre NPCs, propagação de informação, comportamento emergente, videojogos, comunicação

credível

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

NPC	Non Player Character
RPG	Role Playing Game
NCM	NPC Communication Model
GEQ	Game Experience Questionnaire
QDG	Quest Dependency Graph
AI	Artificial Intelligence

1 Introduction

Contents

Knowledge about the state of the world and what the player is doing on that world are very important requirements for any NPC to function properly. By enabling NPCs to share knowledge between themselves about the state of the world, over time they will change their perception about the world around them. One of the ways in which the behavior of NPCs can change is in role playing games (RPGs).

In RPGs, for example, NPCs that are able to give quests to the player require some type of knowledge about the world or the player in order to know if they should give the player the quest they hold. Traditional role playing games (RPGs) have always used some sort of quest based system to tell their story. As the player explores the world, he interacts with different NPCs with the ability to give the player the quests he needs in order to advance in the game.

There are multiple ways by which NPCs can access information about the world. Through global databases that store data about the world and the player or multiple smaller databases that store data in different categories. Sometimes an NPC only needs to keep track of a specific action and check if that action has already happened or not in order for him to function properly.

Some games try to make "smarter" NPCs by improving their behavior and the way they interact with the world and the player. Some games like Middle-earth: Shadow of Mordor [1] go as far as procedurally generating NPCs with unique personalities in a system known as *Nemesis System*, which allows the artificial intelligence of non-playable characters to remember the deaths of the game protagonist and react accordingly.

Before we get to the problem, we first need to define what we mean by "believable communication"

1.1 Believable Communication

In a simulated world environment we can change and adapt rules as we please that would not be possible to apply on a real world situation. I can, for example, decide that the world I am creating is void of gravity, and that trees grow from top to bottom and not the other way around.

In games of the genre role-playing, we are playing a role. And as we play that role we usually abide by the rules of that world. When those rules are different from the rules people are used to follow, the player has to learn those new rules. By making communication work more like what happens in reality, the way NPCs communicate with each other becomes easier for the player to understand.

There is a good reason some RPG games try to use reality as a model for mapping the behavior of the game world. The theme of the game changes, and the events can take place in the past, present or future, but most of the rules present in the game have some type of truthfulness that comes from reality.

What makes the game "believable" is the ability of the player to recognize something in the game that is present in reality. The less believable a game is, the more a player is forced to learn the world

[1] https://procedural-generation.tumblr.com/post/131491361090/middle-earth-shadows-of-mordor-2014-the-open

rules, because when he perform an action he does not know what the outcome might be.

When trying to make a game more believable we are trying to model the world in a way that most players would agree with intuitively, without the need to understand first how the world they are in behaves.

Let's take, for example, a scenario where $PersonA$ is sharing a secret with $PersonB$. In a believable scenario, when these two people share a secret with one another, only both of them would know what the secret is.In a less believable scenario, when $PersonA$ shares with $PersonB$ the secret, $PersonC$ on the other side of the world also receives the secret when $PersonA$ and $PersonB$ finish sharing it.

A player without any previous knowledge about the game would expect, based on what he knows about reality, that only $PersonA$ and $PersonB$ know what they talked about. If that doesn't happen, the player has to learn about what happens in that world when he sees two NPCs talking with each other. This learned behavior is not necessarily bad, but it can lead to some confusion from the player and he might eventually feel disconnected from the world he is experiencing as that world becomes more different from the reality the player is used to.

Another example of believability, especially within the range of known possibility or probability, would be the way in which people interact with one another and the world around them, and in my hypothesis in particular, the way NPCs communicate with each other.

1.2 Problem

Communication in current games is not believable when it comes to actually sharing information between NPCs since an event that happened in the world is globally known instantly by all NPCs.

Communication between NPCs in RPG games is scarce and usually exists to make the world seem more alive to the player by making the player watch how NPCs interact with each other. The participants in the conversation are not actually changing the state of the world and the fact that they are talking to each other is used to convey to the player the idea that they are related in some way like friends, family or even enemies talking before a fight.

Communication can also happen between NPCs as a way to tell a story to the player, and can happen as a video cut scene or a real time event where the player sees the NPCs reaching one another and talking about something that is relevant not to them but to the player.

Communication between NPCs is a tool that is useful for the player and not for the NPCs themselves. When NPCs talk to each other, the end goal is usually to help the player progress in the story or to make the world more realistic.

One thing traditional RPGs do very well is to keep the story they are trying to tell consistent. The player is free to explore the world and to change it to a lesser degree. Some events can be procedurally generated and a story can have different endings based on the choices a player makes.

The problem arises when the player changes the state of the world. Some NPCs require the player to perform certain actions in order to complete a quest and other NPCs might only give certain quests to the player if he has all the requirements for that quest. The common thing among NPCs is that they must have some knowledge about the state of the world in order for them to do the things they are supposed to do, being those things related to the interaction they have with the player, or more simple behaviors like changing their movement pattern and daily routine.

What we see happening in current RPGs is that since the game changes when the player does some action on the environment, and every NPC has access to the state of the world, he knows about the actions performed by the player the moment that action takes place, no matter who the NPC is or where he is located in the world. In games like World of Warcraft, when the player completes a quest in *Location A*, no matter the distance, another NPC in *Location B* can change his behavior and possibly unlock a new quest for the player to accept.

Usually in RPGs, When the player changes the state of the world (being that because he chopped down a tree or saved someone from drowning) the update on the game world is felt by all the NPCs that depend on those actions for their behavior, and it usually happens instantly. The fact that a change in the world is known instantly across all the NPCs for whom that world change is relevant presents a problem when it comes down to believability. In a believable world, When people perform certain actions it takes time for more people to know that those actions took place.

The fact that information generated by the world is globally available to all NPCs that inhabit that world constitutes a problem in a world where a more believable communication is desirable.

1.3 Hypothesis

The solution we are proposing is a way for NPCs to share information generated by the player (or the world) between themselves, and as such being able to react and adapt over time to the changes happening in the world.

The sharing of information must not be instantly know to all NPCs that require it and, like in the real world, take time to reach another NPC. In this work we consider how information that is being generated in a simulated world can be shared among NPCs and how this NPCs can propagate information about the world over time.

By allowing the player to observe in real time how his actions are influencing the world around him, we hope to understand what benefits can be drawn from using a system where information does not reach NPCs instantly but is propagated over time.

We will propose a system based on social networks, along with a model of how information can be processed and distributed between agents in a simulated world environment. I will further use this system on a game scenario to test the validity of this model.

1.4 Contributions

There are 4 main contributions with this dissertation:

- The research done on current games that improve how NPCs communicate and behave using different AI techniques.
- The definition and validation of a model for how NPCs communicate with each other.
- The implementation of a working model in a game scenario
- The elaboration of tests with human players, where they were asked to play two different versions of the game in order to determine if our model improves user experience and the player understands how NPCs are communicating.

1.5 Outline

The remainder of this document is divided as follows:

- In Chapter 2, Related Work, we begin by presenting some of the approaches made by current RPGs for improving how a NPC behaves and how quests and events are created. Then I will explore the different types of networks and how these, in turn, can be used as a way of making NPCs more realistic.
- In Chapter 3, NPC Communication Model, we present a detailed analysis of all the parts that make up our model, how all the parts work together to make the model work, and additional parts that could very well have made it into the final model but didn't for the sake of keeping the model simple. I will also go into detail on how someone could change the parameters of our model to reach the behavior he wants the game world to have.
- In Chapter 4, Implementation, we talk about the development of the tools used during the multiple iterations our our model using the Unity Game Engine [2]. We then talk about the development of a game prototype, built from the ground up using those tools, and used for player testing, where our model was implemented as the core game mechanic.

[2]https://unity3d.com/

- In Chapter 5, Evaluation, we present the results of our the tests with human players, the impact our model has on the game and how beneficial is this model to the overall gaming experience of the player.
- In Chapter 6 we conclude this dissertation with a summary of the work presented and future work, since this base model can always increase in complexity.

2

Related Work

Contents

Quests are one of the most important tools a designer has to motivate the player to do a certain action in the game. Quests have the ability to direct the player to certain locations and to tell a story to the player as he completes the challenges presented to him.

Communication between NPCs can play an important role in determining the quests given to the player since the fact that two NPCs can communicate changes the way they see the world and as a result the type of quests they want to give to the player.

Most RPG games implement a quest system as an algorithm that walks a graph. The quests themselves, if they are not procedurally-generated, are stored in some uniquely-identifiable form that indicates what prerequisites it has and whatever other requirements are needed for a player to accept or begin the quest.

In this section I will first start with a brief description of what constitutes a Quest followed by the Quest Dependency Graph (QDG) systems used by NPCs in many of the current RPGs quest systems.

Then I will talk about different techniques used in different games to make them more realistic and engaging for the player. We will see how all this techniques and NPC behavior changes can lead to different types of gameplay. It is our hope that the change in the ways different NPCs communicate can also lead to a change in gameplay, making it more engaging and diverse.

We end this chapter with an in-depth description of the various types of networks and how we will be using the behavior of those networks to achieve a more believable communication between NPCs. This network based approach to communication could be used as a replacement tool for the standard QDG system present in RPG games like World of Warcraft (WOW)[1].

2.1 Quests

Quests in role-playing games can be described as tasks the player has to complete (alone or in a group) in order to gain some type of reward. When trying to complete a quest, the player interacts with the games world to unravel the quest or understand the plot behind the quest[2].

A quest is a “hunt for a specific outcome”, in contrast to simply winning a game [3]. Quests could be understood as tools used in role-playing games to avoid putting players in a position where they only perform a repetitive action (also known as *grinding*).

One of the most important components of a quest system present in RPGs are *quest chains*. They could be understood as a group of quests that must be completed in a specific sequence. They are one of the mechanisms used to disclose the plot (or storyline) to the player, and what gives the player actions meaning.

[1]World of Warcraft, Blizzard Entertainment, 2004

[2]https://www.gdcvault.com/play/1019964/Effective-Quest-Design-in-MMORPG

It is interesting to understand how the communication between different NPCs could change the type of quests they want to give to the player. We will discuss this type of situations in the next chapters.

2.1.1 Quest Dependency Graph

A graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense “related”. The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called an arc or line) [4].

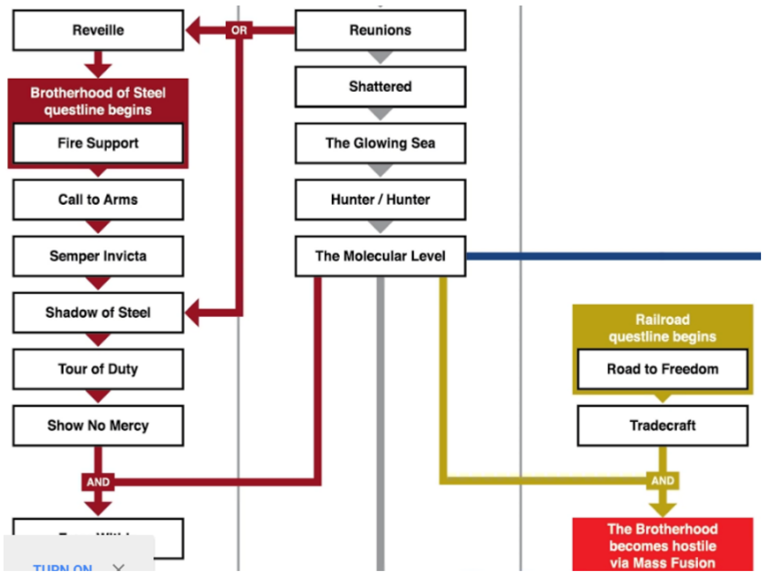


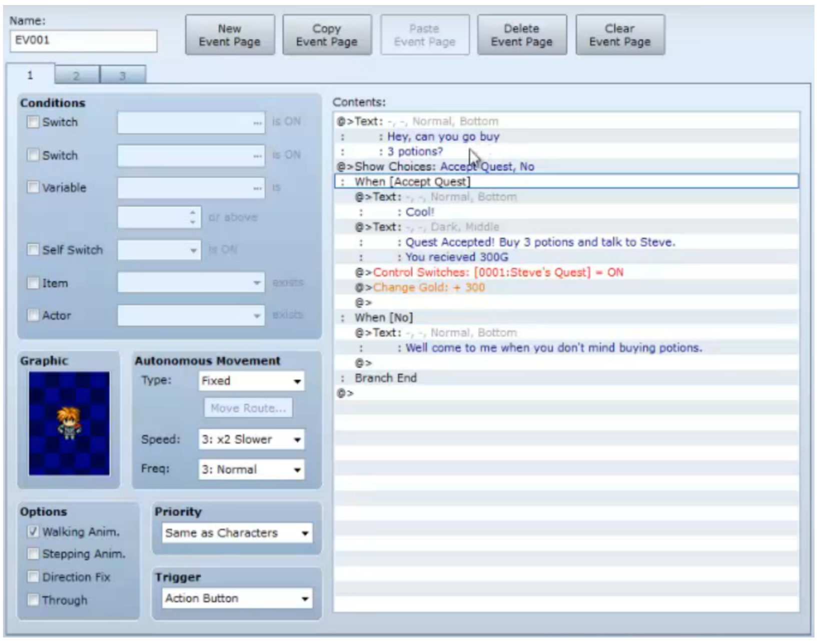
Figure 2.1: This image, created by the Reddit user Th3Element05Image, describes the flowchart of the Fallout 4 quest tree. Image adapted from `https://gamerant.com/fallout-4-mission-tree`.

Quests can follow a path through the story plot, and quest dependency graphs are directed graphs, as they represent not only the quests to be given to the player, but the prerequisites for that quest to be given.

Modeling quests through dependencies is very common and still used in most RPG games. Engines like RPG Maker have a way of representing quests through dependencies, allowing the creation of *quest chains*.

One advantage of quest dependency graphs and quest lines is how easy it is to create simple stories in a small amount of time. It also allows the designer to have better control over the story without “breaking the game”. If quests are created procedurally instead of using a graph systems like this one, a game can end up with a storyline that does not make sense.

Games like ’Middle-earth: Shadow of Mordor’ use a mission based system to progress through the story of the game but also allow NPCs to give players secondary procedurally generated quests. This quests have a lesser impact on the story but give the game more replayability.


Name: EV001

Conditions: Switch [ ] · Switch [ ] · Variable [ ] · Self Switch [ ] · Item [ ] · Actor [ ]

Autonomous Movement — Type: Fixed · Speed: 3: x2 Slower · Freq: 3: Normal

Options: [✓] Walking Anim. · [ ] Stepping Anim. · [ ] Direction Fix · [ ] Through

Priority: Same as Characters

Trigger: Action Button

Contents:

```
@>Text: -, -, Normal, Bottom
:       : Hey, can you go buy
:       : 3 potions?
@>Show Choices: Accept Quest, No
:  When [Accept Quest]
   @>Text: -, -, Normal, Bottom
    :       : Cool!
   @>Text: -, -, Dark, Middle
    :       : Quest Accepted! Buy 3 potions and talk to Steve.
    :       : You recieved 300G
   @>Control Switches: [0001:Steve's Quest] = ON
   @>Change Gold: + 300
   @>
:  When [No]
   @>Text: -, -, Normal, Bottom
    :       : Well come to me when you don't mind buying potions.
   @>
:  Branch End
@>
```

Figure 2.2: Event window in RPG Maker. Here we can see how RPG Maker allows users to create quests for NPCs. Image from video https://www.youtube.com/watch?v=cS7AuM1F5kg

2.2 Radiant A.I.

Radiant A.I. [3] is an AI system present in the game The Elder Scrolls IV: Oblivion[4] that gives NPCs the ability to make their own choices based on the world around them. They'll decide where to eat or who to talk to and what they'll say. They'll sleep, go to church, and even steal items, all based on their individual characteristics.

Within the Radiant AI system, Bethesda encoded the idea of quests that the player can pick up at inns throughout the province. One of its formulae is: "$\{NPC\}$ has been captured, and is being held at $\{Location\}$." This quests are called *Radiant Quests* and they randomly select the goal or target location based on the player past actions.

This system is a step in the right direction in making RPG games more engaging and personal. The Radiant A.I. system gives more control to the NPCs and makes them more autonomous.

As you perform tasks for NPCs or terrorize them by ransacking their home, the NPCs develop feelings about you. If you're good friends with a particular NPC and barge into his house during the middle of the night, he may offer you lodging rather than demand you leave the premises. If you swing your weapon near an NPC, knock items off their dinner table, or try to steal something of value, they'll react with an appropriate level of hostility given their prior relationship to you.

The relationship between different NPCs is something to consider when trying to make the commu-

[3] http://elderscrolls.wikia.com/wiki/Radiant_A.I..

[4] The Elder Scrolls IV: Oblivion, Bethesda Game Studios, 2006

nication between different NPCs look more believable to the player. It is also important to note that the perception that an NPC has about the player changes over time. As the player interacts with the world the way NPCs interact with the player changes.

2.3 Smart Terrain

The game “The Sims” introduced an architecture known as “smart objects” to the game world. The main thrust was to offload a number of functions from the agent onto the objects in the environment. Each inanimate object in the game (that the Sim can interact with) contains two important categories of information: what benefit the object can provide for the Sim and how the Sim interacts with the object.

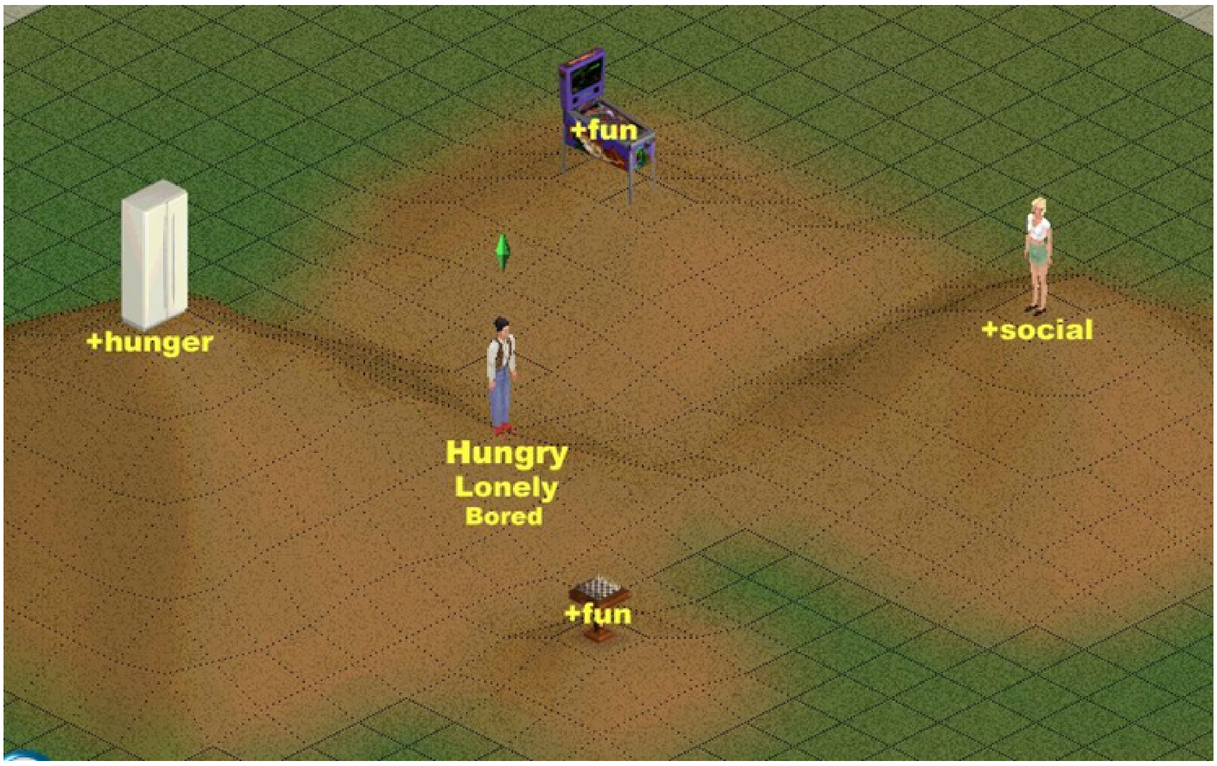


Figure 2.3: “Smart Terrain” in The Sims. AS we can see in this image, each object in the map satisfies one of the NPC necessity. Once that necessity is satisfied, the NPC moves to another object. Image adapted from `http://gameai.com/wiki/index.php?title=The_Sims`.

The behavior used in *Smart Terrain* is a good way of making NPCs in the world behave in a more realistic way. A social network that is based on random change encounters between NPCs has a better change of spreading the correct information to the right NPCs.

The communication between NPCs can lead them to change their goals, and the *Smart Terrain* technique used on *The Sims* could be used to change the movement of the NPCs to different locations they are more interested in, playing an important role in deciding if $NpcA$ crosses the path of $NpcB$, allowing both of them to start a conversation.

As an example, imagine that two NPCs are gathering wood but at a certain point in time a message reaches one of them that the player is killing every tree chopping villager. At that point, the goal that NPC had of gathering wood drops in value and other goals get stronger. As his goals change, so does his behavior and movement. The change in behavior, in this case, was caused by our NPC Communication Model. This message did not reach the second NPC, unfortunately, and because of that he continued chopping trees until he got eventually killed by the player.

2.4 Emergent Gameplay

Emergent gameplay refers to complex situations in video games, board games, or table top role-playing games that emerge from the interaction of relatively simple game mechanics.[5] We can observe emergent gameplay happening when, for example, players are allowed to spawn any object they desire to solve a puzzle, like in the game Scribblenauts [6].

Emergent gameplay can be classified as intentional or unintentional. Unintentional emergent occurs when creative uses of the video game were not intended by the game designers and can happens as a result of glitches, when the player finds other ways to play the game besides the one the designer intended for the player. Virtual economies where items are traded between players, like on the MMORPG game World of Warcraft [7] are also good examples of emergent gameplay.

Intentional emergent gameplay can happen as a result of a game mechanic like the one proposed in this model, where the communication between NPCs can lead to unexpected changes in the behavior of the world and the NPCs living on that world.

Two of the most interesting emergent gameplay situations that happen on games can be seen on the A.I. “The Director” from the game *Left 4 Dead* and the emergent behavior present in the game *Assassin’s Creed Origins*.

2.4.1 The Director

This artificial intelligence used on the game Left 4 Dead (L4D) features a dynamic system for game dramatics, pacing, and difficulty.

Instead of set spawn points for enemies, the Director places enemies in varying positions and numbers based upon each player’s current situation, status, skill, and location, creating a new experience for each play-through. The Director also creates mood and tension with emotional cues such as visual effects, dynamic music and character communication.

[5]http://www.jeuxvideo.com/dossiers/00006203/le-gameplay-emergent.htm
[6]https://www.scribblenauts.com/
[7]https://worldofwarcraft.com/en-us/

The Director goes through a number of ”phases”, relating to the state of the Survivors. First there is a “Build Up”, then there is a “Peak” that usually occurs at the peak of combat and then is follower by a “Relax” stage, where players are allowed to relax a bit.

The Director is always measuring how much pressure each individual Survivor is under by monitoring how many enemies are around the player and how much damage has the player taken.

Using this A.I., the game *L4D* was able to create a richer game experience by dynamically changing what happens in the world based on the player’s actions. Just like in this dissertation, the player’s actions are ultimately what shape the behavior of the game.

The communication between NPCs can lead to a change in the state of the world and this change can offer new challenges for the player to overcome.

2.4.2 Assassin’s Creed Origins

In *Assassin’s Creed Origins*, the NPCs A.I. will interact and react in much more believable ways while living their lives that did not happen on previous games of this franchise. Certain NPCs will remember recent events and, if appropriate, feel stressed out by them.

On a specific example, after a big battle ended and a lot of people died, the commander was carrying so much of stress at this point (because the AI carries stress), that he could not go right back to sleep, so he started roaming his own fort, going to one of the towers and looks like he’s really doing a lookout

The mission system in ’Assassin’s Creed Origins’ was changed from a scripted approach to a goal-oriented one. Giving NPCs more autonomy and yet maintain control over the important narrative-driven gameplay moments is what mainly gives rise to the emergent behavior in this game, changing the way designers create and think about the behavior of NPCs in the world.

2.5 Decentralization

Decentralization is the process of redistributing or dispersing functions, powers, people or things away from a central location or authority [8]. In a decentralized system, lower level components operate on local information to accomplish global goals. The global pattern of behavior is an emergent property of dynamical mechanisms that act upon local components, such as indirect communication, rather than the result of a central ordering influence.

For the implementation of my model, I will be using a type of *peer network data store* as a way of storing information about the events happening in the game world. Each NPC represents a node and has a specific data (memory) attached to it. For as long as the NPC exists in the network, his data can be shared with other NPCs. This is a very important part of the architecture in my model.

[8] https://en.wikiquote.org/wiki/Decentralization

The game itself is not in control of the data produced by the player while he is playing and has limited control over each NPC, who works as an independent agent and his behavior is only influenced by his internal state and not the authority of the game. This can create very interesting situations and gameplay experiences during the game, since not even the designer knows what is going to happen.

One of the most well known examples of a natural decentralized system happens in ant colonies, where ants communicate to other ants where food is, how much food there is, and whether or not they should switch tasks to forage based on cuticular hydrocarbon (CHC) scents and the rate of ant-interaction. The colony captures precise information about the current availability of food and thus whether or not they should switch to foraging behaviour all without being directed by a central controller or even another ant.

Imagining an ant as a single NPC, my model allows for this type of behavior to happen naturally, since NPCs can spread information through each other in a system where $NPC_A \longrightarrow NPC_B \longrightarrow NPC_C$.

2.6 Networks

A network is a system whose elements are somehow connected [5]. The elements of a system are represented as nodes and the connections among interacting elements are known as ties or links. Each node is then capable of sharing resources with the nodes it is connected to through his links.

A quest dependency graph could be described as a network where each node is a quest and ties are represented by the dependencies that exist between quests.

2.6.1 Social Networks

A social network is a social structure made up of a set of social actors (such as individuals or organizations), connection ties, and other social interactions between actors. In an agent-based model, the agents update their internal state through an interaction with their neighbors and the emergent macroscopic behavior of the system is the result of a large number of these interactions. [6].

The study of networks has played a central role in social science, and many of the mathematical and statistical tools used for studying networks have been first developed in sociology. [7]

Social networks can sometimes be organized in groups of agents who share similar properties. This networks form clusters of individuals, where each individual belongs to one or more cluster. Some social networks can then be described as Cluster Networks

2.6.2 Cluster Networks

In general, clustering means grouping unlabeled observations [8]. The idea with clustering is to find natural groups of items, which are in some way similar to each other and share some common properties. In social networks, clusters are often called communities.

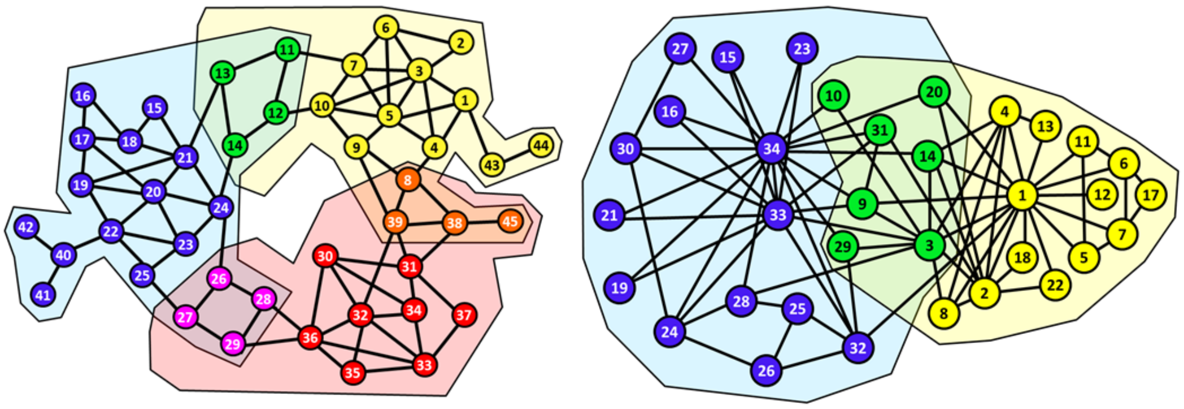


Figure 2.4: Representation of different overlapping communities organized in clusters. In both images we can observe how different clusters can intersect. On the left image we can see multiple cluster intersections. Image from [1]

There are two types of clustering. In *hard clustering*, an item is assigned to just one cluster, while in *soft clustering* each item can belong to multiple clusters with different strengths.

Some social networks have dense groups of nodes and some nodes have disproportionately many connections. The structure emerges, because friendships are not formed randomly. Instead, people tend to become friends with those who are similar to themselves, creating a community.

This is called *homophily* and can be described as the tendency of individuals to associate and bond with those similar to them. My model takes into account the interests of both NPCs to determine what type of message they are going to talk about. A more complex version of my model could take into account the clustering of NPCs by different categories to determine their affinity.

As an example, if two NPCs belong to the cluster 'Female' and to the cluster 'Appletown', their affinity will be higher than another NPC who belongs to 'Male' and 'Georgetown'.

2.6.3 Temporal Networks

Most interactions in a network are not continuous, but have a finite duration. We must therefore view the underlying networks as temporal networks. The concept of temporal networks is an extension of complex networks as a modeling framework to include information on when interactions between nodes happen.

A communication network is a type of time-varying network where each link is relatively short in

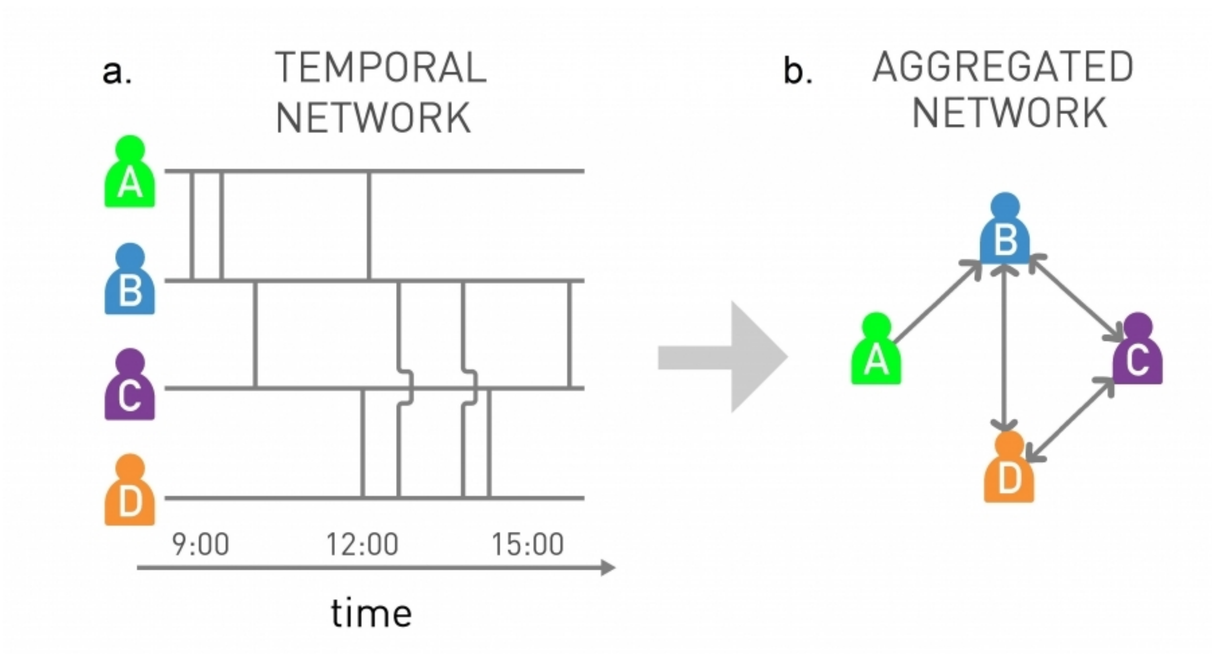


Figure 2.5: Representation of the interactions between agents in a temporal network. On the left image we can observe the transmission of information happening across time. On the right image we can see the network obtained by merging the interactions in the left image. Image from [2]

distance. Time-varying networks are characterized by intermittent activation at the scale of individual links.

A temporal network can be described by the following properties:

• **Reachability** The set of influence of a node *i* is the set of all nodes that can be reached from *i* via time respecting paths, note that it is dependent on the start time *t*. The source set of a node *i* is the set of all nodes that can reach *i* via time respecting paths within a given time interval [9].

• **Latency** Latency is the time-varying equivalent to distance. In a time-varying network any time respecting path has a duration, namely the time it takes to follow that path. The fastest such path between two nodes is the latency, note that it is also dependent on the start time.

Inspired by the propagation of diseases, we can see in Figure 2.5 the timeline of the interactions between four individuals. Each vertical line marks the moment when two individuals come into contact with each other. If $NpcA$ receives some message, that message spreads from $NpcA$ to $NpcB$ and then to $NpcC$, eventually reaching $NpcD$. If, however, $NpcD$ is the first to be infected, the disease will never reach $NpcA$. This is because there is a temporal path between NPCs and this fact is very important to consider on the implementation of my model, because information spreads over time and the time an NPC receives information is crucial to know how that piece of information is gonna propagate through the network.

In our model, the way data is being managed and shared across agents is dependent on time. Since information is not global (i.e. the game itself does not store data related to player actions and events), the time at which an action happens is very important to know how many NPCs know about a certain information and at what time.

Two NPCs are not constantly interacting with one another but instead come into contact with one another at specific events in time so, in the end, their interaction is not continuous and has a finite duration.

It is also important to note that over time NPCs will forget information, as it is implemented on our model and discussed further ahead, so even if an NPC has some information to share, if he takes a long time to interact with another NPC, there is a chance that specific information will get lost.

2.6.4 Weighted Networks

In this type of network, not all ties have the same capacity. In fact, ties are often associated with weights that differentiate them in terms of their strength, intensity, or capacity [10] [11].

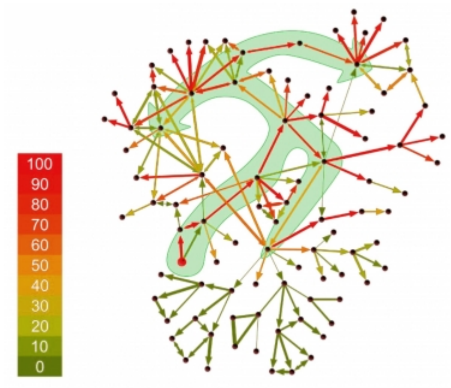


Figure 2.6: Weighted Network describing individuals (nodes) passing information between each other in a real mobile phone network. Image adapted from [2].

Although weighted networks are more difficult to analyze than if ties were simply present or absent, a number of network measures has been proposed for weighted networks, like *node strength* (sum of weights attached to ties belonging to a node) or closeness (using some sort of distance measuring algorithm).

Consider, for example, the diffusion of information in a mobile phone network as seen in Figure 2.6.

The mobile phone network allows us to explore the role of tie strengths and communities on spreading information [12].

Let us assume that at t=0 we provide a randomly selected individual with some key information. The more time two individuals talk, the higher is the chance that they will pass on the information.

Once the information reaches a member of a community, it can rapidly reach all other members of the same community, given the strong ties between them. Yet, as the ties between the communities are weak, the information has difficulty escaping the community. Consequently the rapid invasion of the community is followed by long intervals during which the infection is trapped within a community.

What can be observed here is that there is a reduced speed of transmission on information between communities, indicating that the information is trapped within communities. Indeed strong ties tend to be within communities while weak ties are between them [13].

As you can see by this example, weighted networks can also be organized in clusters, where members of the same cluster tend to have stronger ties between them than with members of another cluster. This behavior can result in the formation of communities.

Inspired by the mobile phone network, we can imagine a scenario where each village forms a community, and where information spreads faster between members of the same village. It is also important to make the distinction between the interaction between members of the same community (stronger interaction) and members of different communities (weaker interaction).

On the final prototype we created 4 different types of communities, where each one of them had a a higher change of being interested in different topics. It was observable that when two NPCs talked, the community they belonged to had a strong impact on what type of information they were interested in talking about.

It was easily observed how a message how a type of message that had greater interest and weight to the community was talked a lot more than a message with less interest and weight.

2.7 Conclusion

In the sections above, we discussed some techniques and tools used in RPGs to change the behavior of NPCs and make them more complex and engaging to the player.

With the *smart terrain* technique we saw how characters made decisions to move in the environment to satisfy their needs and how this behavior could be used in our model to allow for a bigger number of encounters between NPCs and as a result spread information in a way that is not random.

We also got to see how a character could behave more autonomously and not overly constrained by the game rules. This gave the notion of using NPCs as independent agents and to see them not only as characters in a story but also as nodes inside a social network.

Then, with the development of the *Radiant A.I.*, we see how player's actions and experiences could be used to change the behavior of NPCs and how this system gave move replayability to the game, managed to make NPCs more believable and increased the personal relation the player develops with the world and its characters. The *Radiant A.I.* system also shows us how quests could be smartly constructed to direct the player to a certain location without the player feeling like it was forced to go to that location to begin with.

We talked about the importance of *quests* and how they can be generated as the result of sharing information across networks of NPCs. It is important to note that quests are not necessarily the only way in which we can use the information that is being passed around in the network. We could very well change how the NPC behaves towards the player or how he moves around in the world.

In the player testing scenario we are not generating different quests, but making certain types of NPCs react to the player when certain types of information are received. Since NPCs work as nodes on a network, eliminating one NPC can cause that network to be broken and alter its structure. It is important to note that this network is not fixed in shape and size and can change and adapt, breaking or forming new connections over time.

We then explored in depth the different types of networks and how they could be used in our model. The behavior of the agents in the network is one of the most important factor in determining how communication will happen and what kind of information will be shared.

From the concept of social networks we arrive at the term *social actors* and *connections*. We can see how the internal state of agents change when they interact with their neighbors and how similar actors in a network can form *clusters*.

The notion of temporal networks is quite easy to understand when we think about how we interact with people every day. The notion that, for a message to be transmitted, there must be not only a "what" but also a "when". In computer networks the transmission of information is almost instantaneous but, when talking about social interactions, one must consider that information takes time to spread across a network and that is a very important element to also consider.

Weighted networks are very important in modeling how communication happens between NPCs .To solve the problem of making NPC characters behave more realistically, we must consider that not all relations between characters are the same. The type of information being shared between NPCs is measured by weights and each new information will have a different level of importance to each NPC. Ultimately, it is the information being shared between NPCs in the network that will have the most impact in determining how information propagates in the world.

The NPC Communication Model I propose makes use of certain properties of weighted networks, temporal networks and social networks to create a way for NPCs to share information over time.

3

NPC Communication Model

Contents

Our goal is to implement a system that allows NPCs to communicate with one another in a way that makes sense and can mimic the behavior that happens naturally on real life conversations. With this behavior we hope to give to the player a better gaming experience.

Information is shared inside an environment between non-player characters, making them more realistic and relevant to the player as he interacts with the environment over time. It is our hope that the player can see, interact and exploit the communication happening between NPCs to his advantage, and that he can understand the similarities between how NPCs communicate and how people talk to each other.

The game world consists of different types of objects with different types of behavior:

1. *NPCs* that share information with each another over time.

2. *Entities* that are able to perceive events happening on the world and create some sort of information about that event.

3. *Player* who acts upon the world, creating some sort of information over time.

When the player(or NPC) does an action on the world he creates an event. This event generates information under the form of a message. Before we define in detail the different parts that constitute our model we must first understand how they work together.

3.1 Model Definition

Lets first consider all the steps that happen when we create and share messages in the world in order to further define what is required to be present in the final description of this model:

1. There is an event happening in the world. This event can be caused by the player over an $entity$, by that $entity$ itself or even by the actions of NPC_A over NPC_B.

2. An entity creates a message when an event happens to that entity. An entity must have mapped all the possible messages it can generate as a result of different actions done to it.

3. The message generated is spread to all NPCs close enough to the $entity$ to see it or hear it. Only sending messages to NPCs that are in perception distance can be an option here.

4. NPCs evaluate the message received and decide if they want to keep it or discard it. Since there is no reason not to keep a message if they have memory space available, the message received will only be evaluated if there is no more space.

5. NPCs must have a desire to send and receive messages. An assertiveness value measure how much they want to send a message and a cooperativeness value measures how much they want to receive a message.
6. When two NPCs cross paths, there are a few things that must happen in order for them to start a conversation:
   (a) They first see if they are willing to talk and the other NPC to listen (or the other way around)
   (b) They then check if they have any information to talk about between themselves that is of any interest to both of them.
   (c) If there is nothing for them to talk about they ignore each other and continue on their path.
   (d) If there is some message to talk about and all conditions for them to talk are satisfied, they stop their current behavior and start talking with each other.
7. The message received starts increasing in decay over time, becoming less relevant.At this point two different things can happen now to that message:
   (a) The message decays so much that is lost from memory. At this point the message is removed to allow for new ones to get inserted without the need to evaluate them.
   (b) If $NPCs$ talks about that message again, that message is reinforced and the decay resets to the same value as a new message.
8. The cycle repeats itself, as new event happen in the world again.

This model can be subdivided in two distinct activity diagrams. On *Figure 3.1* we can see how a message is generated and reaches an NPC from the moment the event happens and finally reaches the player.

On *Figure 3.2* we see how the NPC behaves, how he interacts with other NPCs and what does it take for two NPCs to start talking. We can also see here what happens after two NPCs talk to each other and what are the conditions for their communication to end.

3.2 Model Components

To describe our model we first must describe all the parts that constitute this model and so we can later see how those parts are relevant and interact with one another.

To evaluate what information is going to be shared and who is sharing that piece of information with who we must first determine what message is the best message to talk about, as we can see in the Algorithm 3.1.

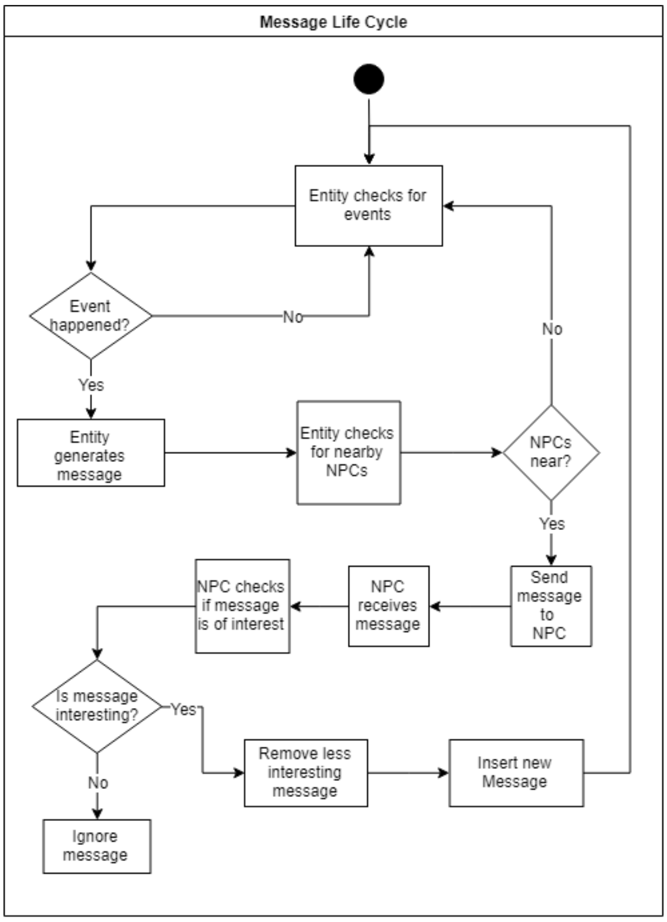


Figure 3.1: Diagram showing the message life cycle. In this diagram we can see steps 1, 2, 3 and 4.

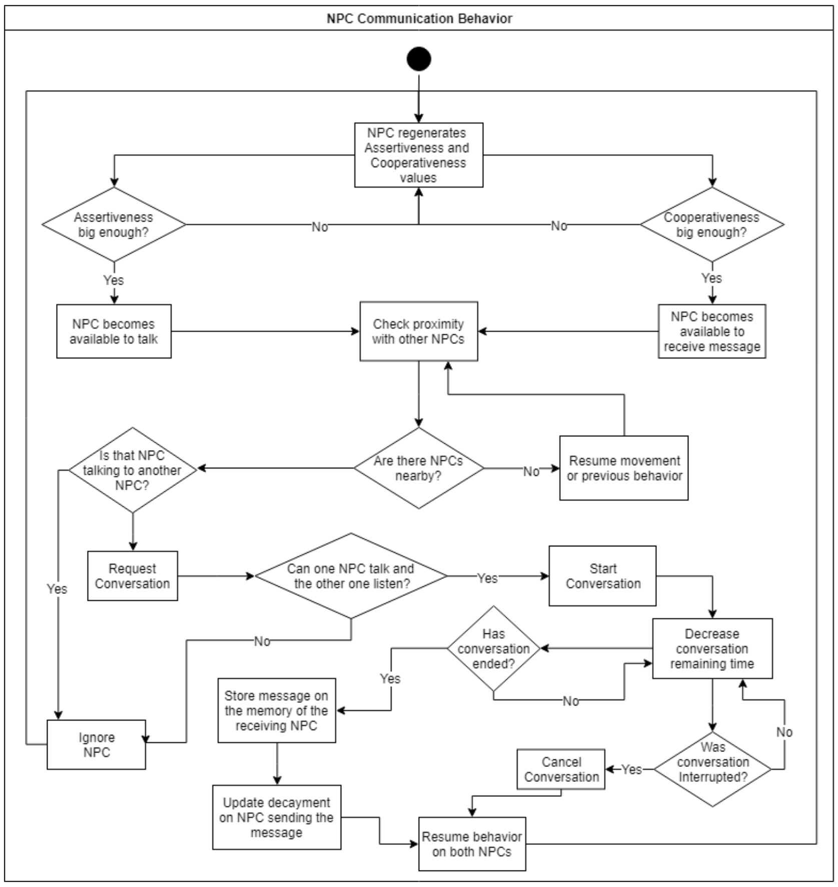
3.2.1 Message

When NPCs talk to each other there is information being passed to one another. This type of information does not only have the actual information being exchanged about but also has additional attributes.

In a real world scenario, this attributes could be the duration of the conversation, the tone by which that information was said, or even the way the person was expressing herself while she was saying what she was saying. The way how we store the information in our brain is also important to take note.

It is not the focus of this dissertation to focus on what NPCs do with the information they receive, but it is very important to know what information they are receiving and how can they share it with other NPCs in a way that makes sense and does not break the game or confuses the player.

The structure of our information will hereby be referred to as *Message*. Each event results in a message being created and each message will have a description of that event, the *time* it takes to communicate that message, how “old” is the information being passed, in a term we call *decay*, and a unique identifier (or *ID*) for the event, so when NPCs talk to each other they know exactly to what event they are referring to when they talk to each other



NPC Communication Behavior
NPC regenerates Assertiveness and Cooperativeness values
Assertiveness big enough?
Cooperativeness big enough?
No
No
Yes
Yes
NPC becomes available to talk
Check proximity with other NPCs
NPC becomes available to receive message
Is that NPC talking to another NPC?
Are there NPCs nearby?
No
Resume movement or previous behavior
Yes
Request Conversation
Can one NPC talk and the other one listen?
Yes
Start Conversation
No
Has conversation ended?
Decrease conversation remaining time
Yes
No
Store message on the memory of the receiving NPC
Ignore NPC
No
Was conversation Interrupted?
Cancel Conversation
Yes
Update decayment on NPC sending the message
Resume behavior on both NPCs

Figure 3.2: Diagram showing how NPCs interact and share messages with one another. In this diagram we can see steps 6 and 7.

Lets take as an example the following event: "When I was walking home the other day I saw Bob climbing a tree and picking an apple. He then ate that apple". This event can be converted in a message, as described on the Table 3.1.

Table 3.1: Example of a Message packet being transmitted between two NPCs

	Data
ID	73
Transmission Time	5
Decay	0.7
TAGS	$\{CLIMB = 9, TREE = 12, APPLE = 5, EAT = 6\}$
Description	Bob was climbing a tree and picking an apple. He then ate that apple

3.2.1.A ID

When the player does an action in the world, that action translates into an event. That event is unique and even though a future event can happen that is exactly the same as the previous event, it will never be the same event.

A message is generated when an event happens in the world and at that moment an unique ID is

Algorithm 3.1: Get Best Message To Talk

```
begin
    $mostAtractiveMessageScore = 0$
    if $npcAAssertiveness > minAssertivenessValueForTalk$ and
     $npcBCooperativeness > minCooperativenessValueForTalk$ then
        foreach $M1 \in MessagesInNpcA$ do
            $scoreA \longleftarrow 0$
            $scoreB \longleftarrow 0$
            $messageScore \longleftarrow 0$
            foreach $tag \in TagsInMessageM1$ do
                foreach $interestA \in NpcAInterests$ do
                    if $interestName = tagName$ then
                        $scoreA+ = WeightInterestA * tagWeightM1$
            $scoreA \longleftarrow scoreA * DecayMessageM1$
            foreach $tag \in TagsInMessageM1$ do
                foreach $interestB \in NpcBInterests$ do
                    if $interestName = tagName$ then
                        $scoreB+ = WeightInterestB * tagWeightM1$
            if $NpcBHasMessage$(M1) then
                $scoreB = scoreB * messageInNpcBDecayment$
            if $scoreB == 0$ then
                $messageScore = 0$
            else
                $messageScore = scoreA + scoreB$
            if $messageScore > mostAtractiveMessageScore$ then
                $mostAtractiveMessageScore = messageScore$
                $mostAtractiveMessage = M1$
    return $mostAtractiveMessage$
```

attributed to that message. This is the only way different NPCs have of identifying if a message is known to both of them. If both NPCs know a message, that message is considered an old message to both of them (See Algorithm 3.1).

3.2.1.B Transmission Time

When people talk, they usually take time to say what they want to say. People also don't usually want to talk every time. The desire to talk becomes less strong the more people talk and eventually, when they are tired of talking, they stop.

During games, the time it takes to talk about a message should also take time to transmit. It is the responsibility of the designer to know how much time it would take for an NPC to transmit a message, but just as it happens with most people, the more an NPC talks with another NPC, the less inclined he is to talk more afterwards because his desire to talk lowered.

There is no clear way of determining how much time a message would take to transmit to another NPC. It depends only on the type of game being developed and the game designer choices. There could be an additional variable that considers how fast a specific NPC is able to talk (a.k.a *talk skills*). It could depend on the ambient conditions around the NPC and if he must talk louder or slower.

If the transmission time of a message was instant and the desire of NPCs to talk was never lowered, they would be able to keep talking forever without stopping and that would be a game breaker.

3.2.1.C Decay

After a message is received by an NPC, it starts losing value. This value is called the $Decay$ of the message. Messages lose value over time because the $Decay$ of a message increases over time. Without $Decay$ there would be no prioritization of new messages over old ones.

It is important for NPCs to be open to receiving new information and share it in the network even if they find that piece of information less interesting than an old message they once heard. $Decay$ helps NPCs prioritize what message should be exchanged based on how old that message was received by that specific NPC.

it is important to note that a message that may be old to NPC_A but may not be old to NPC_B, since a message relating to the same event could have been received during different conversations.

3.2.2 TAG

Trying to understand what type of information is being shared in the world is close to impossible, since every conversation can be different, can be interpreted in different ways and can lead to different people saying the same thing but with different meanings.

To work around this problem we first had to convert the infinite amount of conversation possibilities into a working classification of information. To something every NPC can understand.

So we had to convert our conversations into categories. This categories can, of course, be subdivided into as many subcategories as we want. They have the power to describe in detail all the conversations we can possibly have between NPCs in a way that is usable and cost efficient in a game that can have hundreds (if not thousands) of NPCs. We call this attributes of communication TAGS.

So every time an NPC is saying something to another NPC, what they are really talking about translates into are a ground of TAGS that describe the type of communication they are having with one another.

TAGS are mapped, by default, by the designer of the game when he creates conversations for his NPCs to have. That being said, I can imagine a designer making a piece of software that can recognize specific words in a text and convert it to usable TAGS. It is however advisable, for the sake of clarity and cohesion of a game, that when the game designer creates a piece of conversation, he creates the appropriate TAGS to go along with it.

AS an example, a conversation like ”When I was walking home the other day I saw Bob climbing a tree and picking an apple. He then ate that apple.” could translate to : $TAGS = \{CLIMB, TREE, APPLE, EAT\}$

3.2.3 Weight

It is not only important to know what TAGs constitute a specific message, but also the weight those tags have on the message. The weight of the TAGs is a good indicator of the importance of a message. Lets take a look at the example of Table 3.2.

Table 3.2: Example of similar messages with different degrees of importance

	Message 1	Message 2
ID	73	46
Transmission Time	5	10
Decay	0.7	0.3
TAGS	$\{KILL = 20, CAT = 5\}$	$\{KILL = 20, KING = 500\}$
Description	I saw the player killing a cat	I saw the player killing the King

As we can see, the actions done by the player were very similar. He was caught in the act of killing. The big difference here is that in the case of $Message_1$ it was a cat who died and in the case of $Message_2$ it was the king who died. Now there are a couple of extra conditions to determine what message is going to be exchanged. If the other NPC didn’t have any of these two messages,there was not a big difference in decay between both messages and the NPCs engaged on the communication were not big cat lovers, $Message_2$ would be the one they talked about.

It is important to note that the weight of the TAGs of a message are directly influenced by the interests of the NPC (see Algorithm 3.1), and this interests can some times override the basic importance of a message in favor of a different message they are a lot more interested in talking about.

3.2.4 NPC

Although we already talked extensively about NPCs in the chapters above and their role as agents in a network of NPCs that share messages between themselves, there are still some properties NPCs have that should be addressed here.

3.2.4.A Memory

To make this model work we can’t allow NPCs to have unlimited memory. The more memory an NPC stores for himself, the more computational power it takes to decide what message to share with other NPCs. For the sake of speed and simplicity, we limit the ability each NPC has to store messages.

3.2.4.B Forgetfulness

As a consequence of having limited memory, NPCs will also have to choose what message is more important for them to retain. When an NPC is trying to communicate a message to another NPC, it is very important to know if the new message is of more interest to the NPC than the messages he currently has.

If a new message that reaches an NPC is more interesting than a message that is currently on memory (Algorithm 3.1), the NPC will remove the less interesting message to make space for the new and more interesting message to be recorded.

It is also important to note in Algorithm 3.1 that the decay of a message plays an important role in determining what message is going to be forgotten. As a message becomes older they lose value and new messages that would not be as interesting to the NPC as the old messages become more important for the simple fact that they are novelty.

Algorithm 3.2: Is Message Of Interest

```
begin
    $mostAtractiveMessageScore = 0$
    if $MessageCountNpc < messageMemoryLimit$ then
        return True
    else
        $receivedMessageScore \Longleftarrow 0$
        $lessInterestingMessageScore \Longleftarrow 0$
        $lessInterestingMessage \Longleftarrow nullMessage$
        foreach $tag \in msgBeingTestedTags$ do
            foreach $interest \in npcInterests$ do
                if $interestName = tagName$ then
                    $receivedMessageScore+ = interestWeight * tagWeight$
        foreach $M \in MessagesInNpc$ do
            $totalScoreOfAMessage \longleftarrow 0$
            foreach $tag \in messageTags$ do
                foreach $interest \in npcInterests$ do
                    if $interestName = tagName$ then
                        $totalScoreOfAMessage+ = interestWeight * tagWeight$
            $totalScoreOfAMessage = totalScoreOfAMessage * messageMDecay$
            if $totalScoreOfAMessage < lessInterestingMessageScore$ then
                $lessInterestingMessage = M$
                $lessInterestingMessageScore = totalScoreOfAMessage$
        if $receivedMessageScore > lessInterestingMessageScore$ then
            if $NpcDoesNothaveMessage(msgBeingTested)$ then
                $RemoveMessage(lessInterestingMessage)$
            return True
    return False
```

3.2.4.C Movement

Movement in NPCs can be achieved in multiple ways and even though it is not the focus of this dissertation, it plays an important part in determining when two NPCs cross paths and are able to share a message between themselves.

It was not possible on the final testing prototype to use a goal oriented movement (as talked about before on the *Smart Terrain* section) and it is my belief that a goal oriented behavior for NPC movement would make the transmission of messages more realistic.

On a patrol movement, since NPCs have a specific path to walk during the game, the designer would have to make sure that events happening in $Location_A$ have a chance of reaching $Location_B$, transmitting a message from NPC to NPC until it reaches $Location_B$.

Since message transmission is based on location, the change in movement from a single NPC can have an impact in determining if a message reaches a location or not. It is possible that events happening in $Location_A$ are never able to reach $Location_B$ simply because people living in both this locations never cross paths with one another.

3.2.4.D Behavior and Player Perception

The behavior of an NPC can change during a game. One of the important mechanics this model allows is the fact that with new messages being received, each NPC can reach a different conclusion about who the player is and what he is doing in the world. As such, NPCs in $Location_A$ can have the general idea that the player is a bad person and in $Location_B$ think the player is a good person.

In fact, if the player is able to remove NPCs from the map, he has a chance to break an important chain in the network, breaking the possible communications happening between NPCs. Imagine, for example, that the only way people in $Location_A$ had of knowing what the player was doing in $Location_B$ was through an NPC called 'Bob'. If Bob was killed by the player while moving from $Location_A$ to $Location_B$, every action the player did on $Location_B$ from that moment forward had no way of reaching $Location_A$.

The act of killing Bob will have an influence on the behavior of the world. From that moment forward, the player will not only be unable to interact with Bob (because he is dead) but also every action that the player will do from that moment forward on $Location_A$ have no way of reaching $Location_B$. As a result of this, the perception NPCs will have about the player on $Location_B$ will not change as a result of the player some actions on $Location_A$.

The changes happening to NPCs caused by the sharing of messages through the network is the most important point of this dissertation. A change in behavior happening in NPCs is directly caused by a change in the way NPCs perceive the player and this change happens because NPCs are communicating with each other.

3.2.4.E Assertiveness and Cooperativeness

A conversation can happen both ways. $Assertiveness$ measures how much an NPC wants to share a message he possesses with another NPC. $Cooperativeness$ measures how much an NPC is willing to receive a message another NPC has to give him.

In Figure 3.3 we can see the 4 possible outcomes that can happen when two NPCs cross paths with one another. If, for example, NPC_A is the one who is sharing a message, the more he talks the less assertive he becomes and as a result his value of $Assertiveness$ diminishes.

If NPC_B on the other hand, is the one receiving a message, the fact that he is cooperating with NPC_A makes his $Cooperativeness$ value diminish also (see Algorithm 3.3).

Algorithm 3.3: Update Assertiveness/Cooperativeness

```
begin
    if $\underline{NpcAIsTalkingWith(NpcB)}$ then
        if $\underline{NpcAIsReceivingMessage()}$ then
            $AssertivenessNpcA \longleftarrow AssertivenessNpcA - \Delta time * AssertivenessDecreaseRate$
            $CooperativenessNpcB \longleftarrow$
              $CooperativenessNpcB - \Delta time * CooperativenessDecreaseRate$
        else
            $CooperativenessNpcA \longleftarrow$
              $CooperativenessNpcA - \Delta time * CooperativenessDecreaseRate$
            $AssertivenessNpcB \longleftarrow AssertivenessNpcB - \Delta time * AssertivenessDecreaseRate$
    else
        $AssertivenessNpcA \longleftarrow AssertivenessNpcA + \Delta time * AssertivenessRegenerationSpeed$
        $CooperativenessNpcA \longleftarrow$
          $CooperativenessNpcA + \Delta time * CooperativenessRegenerationSpeed$
        $AssertivenessNpcB \longleftarrow AssertivenessNpcB + \Delta time * AssertivenessRegenerationSpeed$
        $CooperativenessNpcB \longleftarrow$
          $CooperativenessNpcB + \Delta time * CooperativenessRegenerationSpeed$
```

3.2.4.F Interest

Just has it happens with people, NPCs have different interests from one another. This interests are going to determine how willing they are to receive a new message.

If, for example, an NPC has a strong interested in *WOOD*, he will be more willing to receive a message from another NPC that contains the TAG *WOOD*. However, if he is not interested in the TAGS of a certain message there is a strong possibility that he will prioritize an old message from memory instead of the new message.

If NPC_A doesn't have any interest in the messages NPC_B and NPC_B doesn't have any interest in the messages of NPC_A, they will never send or receive any messages between themselves.

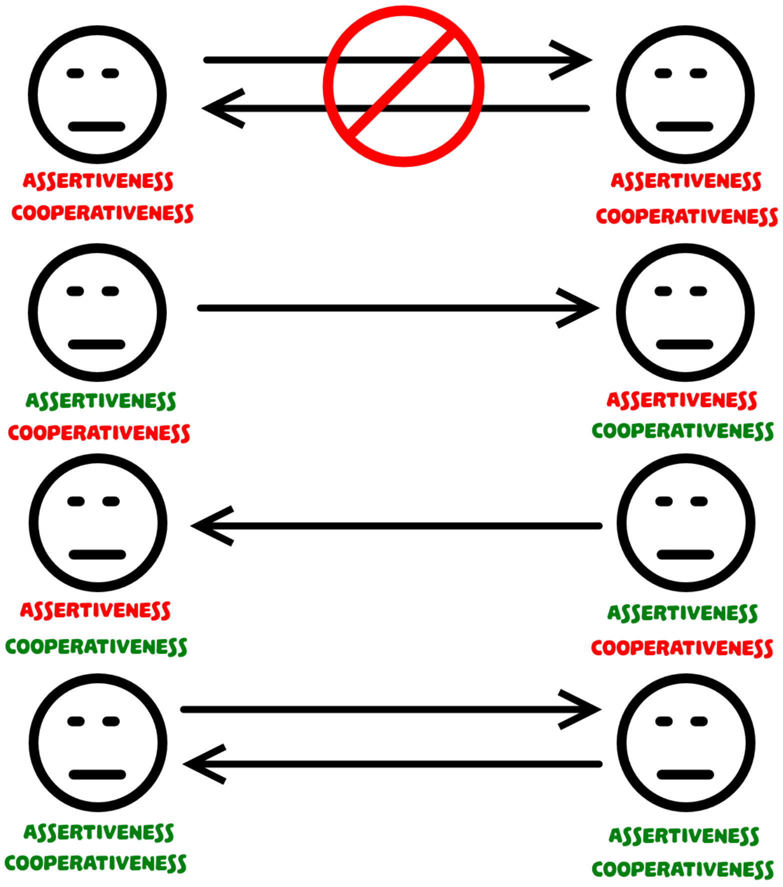


Figure 3.3: Direction of communication happening between two NPCs. Note that There are 16 possible cases for the direction in which NPCs communicate. A message is sent by the NPC who is being $Assertive$ and received by the NPC who is being $Cooperative$

Imagine the scenario where Bob meets $Morty$ as they were walking through the village. This two NPCs have the following interests:

$InterestsBob = \{DESTRUCTION, ROCK, TREES, CACTUS, FENCES\}$

$InterestsMorty = \{GATHERING, CACTUS, BERRIES\}$

Now lets imagine Bob has the following messages:

$MessageA = \{ID : 46, Decay : 0.6, Message :$ *The player destroyed a big rock*$, TAGS : \{DESTRUCTION, ROCK\}$

$MessageB = \{ID : 46, Decay : 0.6, Message :$ *The player killed a horse*$, TAGS : \{KILL, HORSE\}$

In this case, since both players don't have any interest on $MessageB$, it will never be talked about. $MessageA$ will also not be talked about because $Morty$ is not interested in the message.

An NPC will only receive a message if he is interested in the TAGs of that message. So, even if Bob has enough $Assertiveness$ to share $MessageA$ and $Morty$ has enough $Cooperativeness$ to receive it, it will never be talked about because $Morty$ doesn't have any interest that corresponds to $\{DESTRUCTION, ROCK\}$.

The decision for NPCs to only talk about messages they are interested in is a design choice. If the designer wants to enable a message to be talked between all NPCs, he can simply give each NPC a *DEFAULT* interest and give a message that TAG. This way, the message will always be of interest to every NPC.

It is also important to note that even if both NPCs have common interests, if there isn't any TAG present in the interest of both NPCs, that message will never be talked about when they cross paths with each other. In this scenario, both NPCs are interested in $CACTUS$, although no message about cactus is present for them to talk about.

3.2.5 Entity

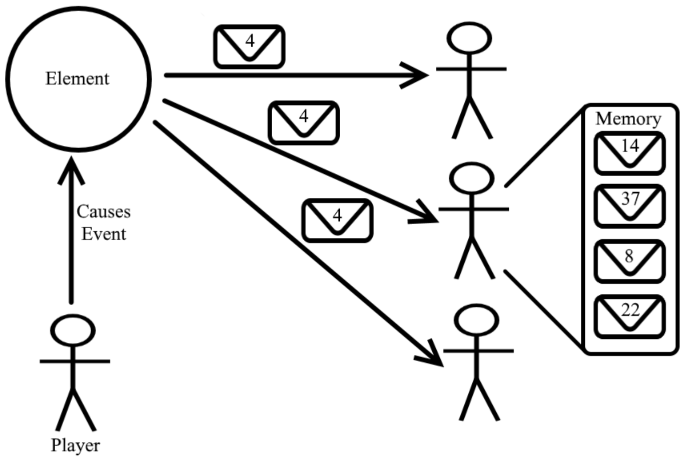


Figure 3.4: Diagram for how a a message spreads from an entity

When an NPC does a certain action in the world he sets in motion an 'event'. This event translates into a message. This message is created by an entity and sent to all NPCs around that entity who are able to *perceive* the event.

Different NPCs can have different a radius of vision. It is the responsibility of the *Entity* to know if, at the moment the event happened and the message was created, a certain NPC was able to see what happened.

When the message is received by the NPC he will not automatically store it in his memory. He will have first to validate that message (see Algorithm 3.2) in order to know know if he will want to keep it.

3.2.6 Events

We consider an event as any action caused by the player (or the world itself) that can be interpreted by an Entity allow him to generate a message as a result. The action of the player cutting down a tree causes an event that will be detected by the tree itself. The tree can then generate a message describing that the fact that is taking damage caused by the player or even that it was cut down.

It is important to note that events are not restricted to player's actions. The simulated world itself can be able to cause events. An NPC can cause an event that is perceived by other NPCs. When they talk they will not be talking about the player but about each other. Even though the player was not involved in this situation, the interaction he will have with NPCs he will talk with from that moment further can change simply because the way NPCs perceive the world has changed.

As an example, imagine the scenario where the player stole an important relic from a nearby temple. Now lets break down what is happening here:

- There are two distinct elements present here: The $Temple$ and the $Relic$. For the sake of simplicity we are only going to consider the $Relic$ entity.
- NPC_A saw the player approaching the $relic$ and stealing it.
- Later that day, the player interacted with NPC_A

In this situation the player wanted to grab a quest from NPC_A and the quest he was going to give the player depended upon NPC_A having a good opinion about the player. Since NPC_A now considers the player to be a thief and a bad person, NPC_A changes it's behavior, and is no longer interested in the help of the player.

In this example we see how a event caused by the player has influenced the decisions of an NPC. What would be more interesting to know is what would happen if NPC_A didn't saw the player stealing the relic, gave the player a quest and discovered while the player was trying to complete that quest that the player was a thief.

You can't blame NPC_A for having an accurate perception about the player, since no other NPC told him what the player was a thief before he accepted the quest given to him by NPC_A.

Now lets imagine the player really wanted to have the quest from NPC_A but since NPC_A received new about the player mischiefs, he will no longer want to cooperate with the player. To change the perception NPC_A has about the player, it could be possible for the player to act as any other NPC, allowing the player to store messages and states about the world and telling NPCs about it.

In this case, the player could do something good (like saving a villager) and telling NPC_A about it, forcing him to change the perception NPC_A has about the player. The player could then be directly involved in shaping the perception NPCs have about the world and eventually change an NPCs behavior.

It is also important to note here that NPCs (and possibly the player) can also be considered entities themselves. If an NPC, for example, kills another NPC, the NPC being killed will generate a message (for example, “I was killed by Bob”) and spread that message with any NPC close enough to have watched it happen.

If, on the other hand, an action is done by an NPC to the player, it is the responsibility of the player to create and send a message to NPCs around him detailing what what happened to him.

In the next section, we will describe two simple and common scenarios where our model is able to create a gameplay experience that was better and more engaging than what normally happens in traditional RPG games.

3.3 Scenario 1

An *NPC* takes into account the *NPC* receiving the message when he decides he starts a new conversation. In this scenario we will observe how an *NPC* changes the message he is saying based on the *NPC* he is giving that message to.

Scenario1: *Bob* was walking through the forest when he notices a sound coming from the trees north of his position. He approached those trees and watched as the player was cutting down a tree. On the way back to the village *Bob* also noticed that the sign pointing to the village on the crossroad was destroyed, probably by the storm from the night before. as he was reaching the first house of the village he stopped to tell *Teddy* about what the player was doing since he knew that $Teddy$ was probably interested in hearing anything related to $Wood$ since he was a Lumberjack. When he passed by the town square he saw $Hope$, and, since she was responsible for the maintenance of the village and the roads, she was interested in anything that was destroyed on her control zone. As a result of all of this, $Teddy$ now thinks that the player must probably be interested in working for him so he has a quest to give to the player. Also, the next time the player passes through the town square, $Hope$ will probably have a request for the player gather some materials, adventure into the woods, reach the crossroad and fix the road sign.

Now lets analyze this scenario:

1. We have a player performing an event on a tree entity and a destroyed sign broadcasting that he was destroyed, probably by the storm the say before.

2. We have two messages reaching an NPC called Bob

3. Bob chooses different messages to talk with the villagers, choosing to talking about the first message with $Teddy$ and the second message with $Hope$

4. $Teddy$ updates his internal perception about the player and $Hope$ updates her perception about the world.

5. Both $Teddy$ and $Hope$ are now able to give quests to the player based on the information they received and share what they learned with the rest of the village.

3.4 Scenario 2

In this scenario we observe how a message can spread in a community and how the world around the player can change based on the player's actions. We can also see how the designer can ensure that a message is able to be exchanged between all NPCs in the community.

Scenario 2: The player kills a villager called $Morty$. $Morty$ is also an entity and when he gets killed he sends a message to all villagers close enough to see the player killing him. The player didn't notice, but behind the bushes near him was Ted, who is a birdwatcher and accidentally watched the whole thing unfold. Ted quickly stops what he is doing and runs back to the village to tell someone.

All NPCs in the village are interested in messages with the tag *IMPORTANT* and this is a message of extreme importance. All the other messages in his memory he could be sharing with his peers are going to stay on hold for now.

The player didn't think anyone saw what he did to $Morty$ and confidently walks into the village. The guards quickly grab him and throw the player in jail. Eventually, as time passed and NPCs stopped talking about that message, every NPC would have removed from memory the message that the player killed someone. In this case what the player did was the talk of the entire town when the player arrived.

Now lets analyze this scenario:

1. We have a player killing $Morty$.
2. This message changes the behavior of Ted, who quickly starts to run to the village to tell someone that $Morty$ is dead and it was the player who killed him.
3. Since Ted got to the village fast and the player didn't come into the village right after he killed $Morty$, there was time for the message "The player killed $Morty$" to spread and reach a lot of people.
4. New messages with a stronger importance would eventually remove this message from memory, since the decay would grow on the message "The player killed $Morty$" as it became less talked about.

3.5 Summary

In this chapter we began by presenting a detailed description of what parts would constitute our model. This description resulted in the subdivision of our model in two distinct parts:

1. How a message is created by the world.
2. How is that message propagated in the world.

We ended this chapter with two different scenarios were we could see our model in action and observe:

1. NPCs adapting their conversations based on who they are talking with.
2. A change in the state of the world over time as a result of our model and the actions done by the player.

4

Implementation

Contents

In this chapter, we present the implementation of the NPC Communication Model on a game made from scratch to test this model.

We start this chapter by describing the game developed where the model is implemented. We will then talk in detail about the multiple tools developed for testing different game scenario and why it was very useful in shaping our model into its final form.

We will briefly analyze the various types of data recorded during development and why they were important when developing the final prototype. We will also discuss how the tools developed were helpful controlling the values obtained during development to guarantee that the behavior of the world made sense in the end.

We will end this chapter by discussing what some of the more common challenges a game designer would face when implementing this model. This challenges are very game dependent and may happen in some games but not others.

4.1 Final Prototype Concept

When deciding how the final map would be, there were multiple things to consider:

1. We needed a map big enough for the player to understand over time the changes on the map. A big map would also allow for the player to explore the surroundings and increase the change of that player to see interesting and emergent behaviors happening in the world.

2. We needed to give the player an objective. Whatever that objective might have been, it needed to take enough time for the player to understand how the world worked but not long enough that he would take a lot of time to complete it and eventually give up. A longer objective to finish would imply that player testing would take a longer time. The average player testing should take an average of 15-20 minutes.

3. Give the player a sense of progression, so he saw the world change as he was playing the game. Making the game more difficult over time was a strategy used to make the game more challenging and keeping the player more focused.

4. Make the world diverse and with enough terrain obstacles to overcome. Here we wanted the player to move through certain parts of the map where it was harder to reach the objective.

5. The player should also be free to explore the map and interact with the environment.

6. Playing the prototype a second time would end up in players making better decisions and the results would be very different that those with players who never tried the prototype before. So we wanted to make sure players got to explore the map

7. Make the player return to places he has been before. If some important event had already happened there, the player should be able to see the changes in its environment. The player should also be able to notice changes caused by his actions in the behavior of NPCs.

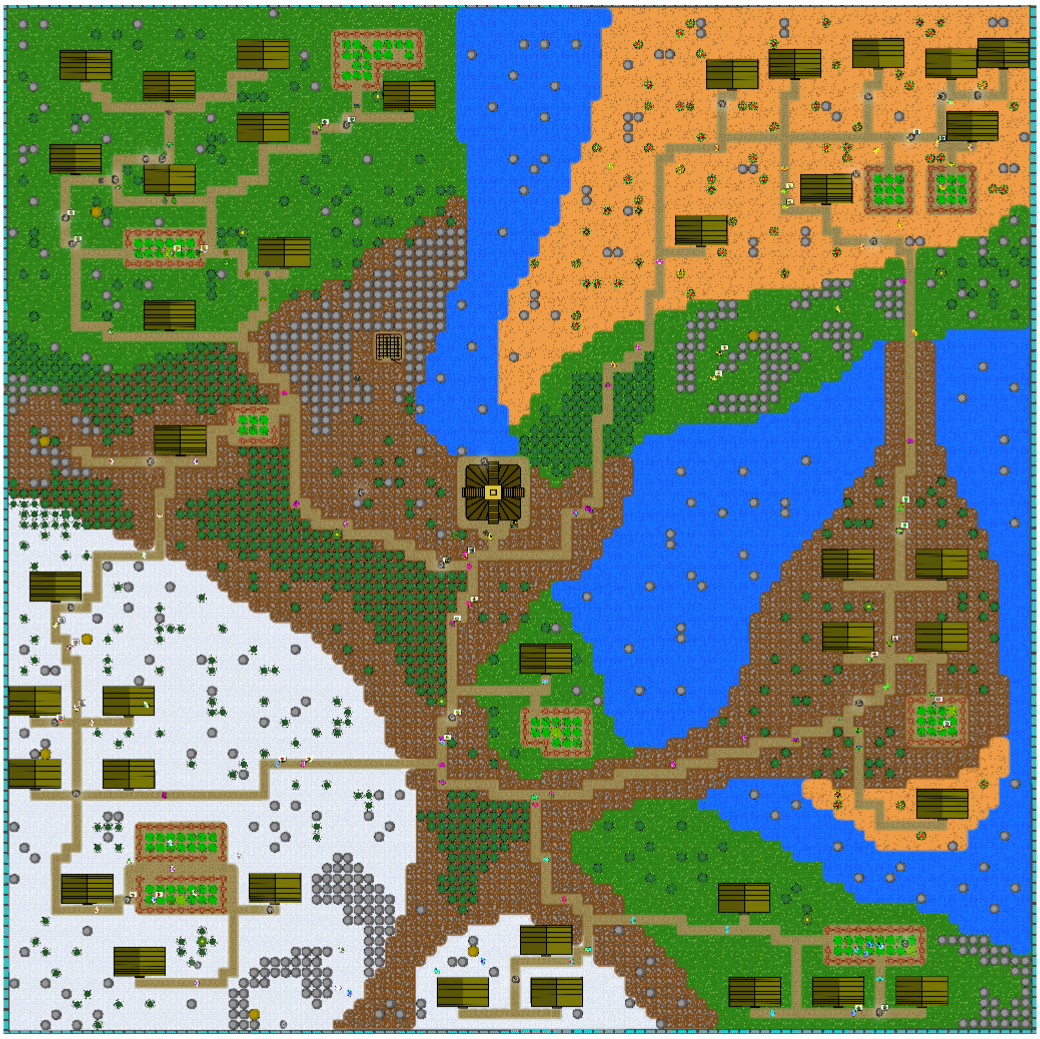


Figure 4.1: Map used in the final prototype. here we can see different communities and roads connecting all of them to each other. Some NPCs wanted through the roads between communities and some stay inside their communities, receiving new messages when a “traveler” comes into town.

4.2 Game Description

When implementing a game prototype with player testing in mind it was necessary to give people who were testing the game something to do. A small story was created, and a map was developed to accommodate that story. There should also be a way to explain to the player how he could control the game, since some of the people testing the game had no previous experience playing this type of games before.

The final decision was to create a game character called “The Prophet”. This character allowed us not only to include a small story detailing the objective the player needed to complete in the world, but also to introduce the controls and abilities the player needed to use in order to complete the objective.

The objective is a simple “Gathering Quest”. The player is asked by “The Prophet” to grab 12 different objects (called ’Relics’) and to bring them to a certain location. Special NPCs called ’Guardians’ protect those objects and will chase the player if he grabs them.

The player is only able to carry 2 relics at a time before he is forced to go back to the center of the map (in a place called “The Temple”) and drop them there.

The choice of limiting the player carrying capacity to 2 objects forces the player to return to certain locations where there is more than 2 objects to collect (like the four distinct villages on the four corners of the map). This gives the player a chance to see what happened on that place after he grabbed those items and how his actions influenced the community.

The difficulty of the game gets bigger the closer the player is of finishing his quest. If the player gets caught by a $GuardianNPC$ he drops all the items he was carrying and respawns in a prison,, where a prison guard lets him go with a warning to ”Be more careful next time!”.

If the player wants to get back the items he lost from getting caught he must return to the place where he was caught. This game mechanic allows the player to see the result of his actions on a specific location since NPCs had a chance to exchange messages between them during that time.

To further illustrate how the gameplay works we created a short gameplay video of the player interaction with our prototype. [1].

4.3 Graphic Choices

When starting to make the game assets it was chosen very early to make a 2D game. The first prototype was made with assets obtained from third party websites and the assets were made using pixel art. They were very simple to use but not easily scalable for making large maps, since it was very time consuming to draw a tilemap using only images as sources.

There was also the problem of getting the necessary assets to populate the game, guaranteeing the assets had the necessary amount of pixels (some assets had 16x16 pixel resolution where others had 32x32 or 64x64).

It was decided the best way to create future maps was to create a terrain generation tool using a different graphical approach and the original Unity Engine project using pixel art graphics was abandoned.

[1] https://www.youtube.com/watch?v=2jzMMSUyrxU

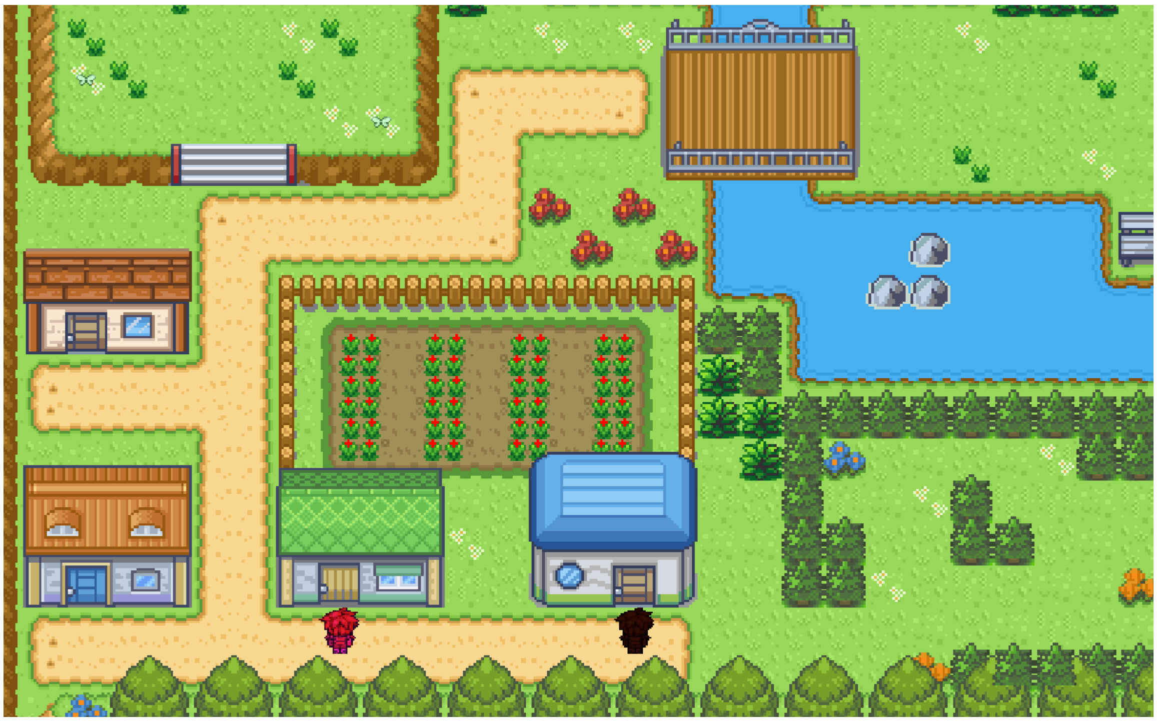


Figure 4.2: Here we can see the first prototype made using third party pixel art assets

4.3.1 Flat Design

In the end the choice of graphics was done primarily because of simplicity, by using minimalism as the main focus of the design. The type of graphics chosen for this project is very similar .

Flat design is a minimalistic modern style of user interface and graphic design, which uses a minimum of elements and excludes any types of complex colors, gradients, highlights and other shiny, textured, shadowed effects [2].

Flat design is very simple, minimalist, and clean. When it comes to drawing assets for games, flat designs really helped us to focus more on the content of the game and behavior of our model and not be distracted by visual effects, something that would probably have happened if pixel art was used instead.

The use of color is another great feature of flat design. Bright and bold colors look attractive and clean. With very few and distinct colors, the game starts to look engaging and visually appealing, creating a proper mood on every possible scenario, being the terrain made of sand, snow or dirt.

The biggest advantage of using flat design was how easy it was to create and add to the game the multiple assets required for the final game prototype and how appealing the game could look in a short amount of time.

Because of the simplicity of *Flat Design* techniques, we were able to draw all the game assets

[2]https://design.tutsplus.com/tutorials/10-top-tips-on-creating-flat-design-graphics–cms-25888

ourselves using a software called Paint.NET [3] and not depend on external sources for game assets.

4.4 Unity Game Engine

There was an early decision do adopt Unity [4] as the main tool to test this model. The reason for this choice was primarily because of how flexible Unity Engine can be and the nature of this work. Not knowing how the model would be like in the end, what type of tools were going to be needed to test the various iterations of the model and how the feedback would be mapped out, Unity was early on the most adaptive tool to satisfy all future requirements.

It was also considered as an option to implement this model on an already existing game, but due to the difficulty in solving future problems on a game not made from scratch to support this model, it was decided that the best way was to develop a simple game from the ground up using the *NPC Communication Model* as the base component.

The NPC Communication Model is described in this project through four different scripts (source code can be found on Git Hub [5]):

1. *Social.cs* is the most important module of the project. It is responsible for determining and process all types of interactions between two NPCs. It determines if NPCs can talk to each other, what is the most important message they have to talk about and if it finds a message to talk about starts the conversation between both NPCs, stopping the previous action those NPCs were doing.

2. *EntityController.cs* controls how entities create and send messages to NPCs.

3. *Message.cs* is the representation of a message

4. *NPCData.cs* stores and manipulates all properties related to an NPC, as well as all the messages in memory.

4.5 Tool Development

There were many tools created for testing this model. Some of these tools were used early on and some tools were used on later iterations. All of them were necessary at one point or another and the final prototype still supports the use of most of them.

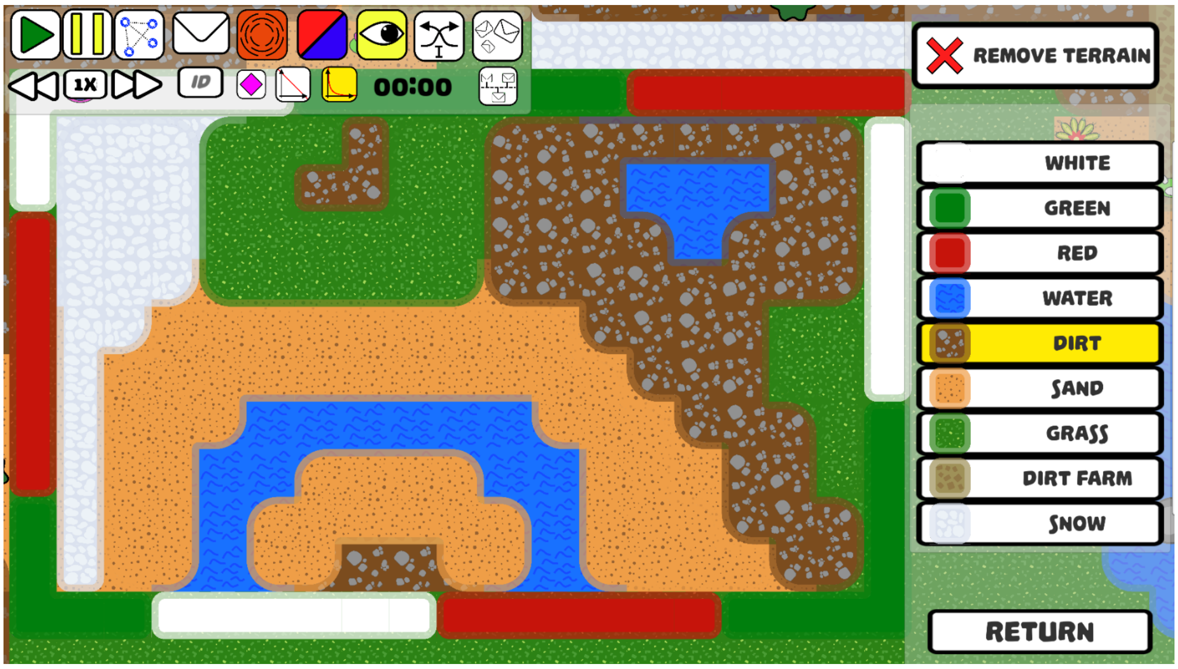


Figure 4.3: The terrain editor drawing multiple types of texture

4.5.1 Terrain Editor

This tools was the first tool developed and was used throughout the entire development of the model. What this tools allowed was to draw a map in real time for usage in specific scenarios. This tool Supported the use of as many base textures as the designer wanted. It was possible to draw two different layers of terrain. The map was able to draw a foreground layer when holding the left mouse button and drawing a background layer when holding the right mouse button.

The ease of use of this tool allowed for some very quick map editing and was probably the single most important tool in saving time and energy when designing different types of simulations

4.5.2 Path Finding and Patrol Movement Feedback

During the early development the movement of NPCs was based on goals. The designer specified where he wanted NPCs to go to and through path finding techniques the game engine moved the player to where he wanted to go. Although this early version was more complex than the final iterations when it came to the movement of NPCs, it was discarded because how computer expensive it was when the number of NPCs reached very high numbers.

On a patrol type of movement, NPCs are give a list of coordinates where they need to move to. When

[3]https://www.getpaint.net/
[4]https://unity3d.com/
[5]https://github.com/Mordokay/ThesisGenesis

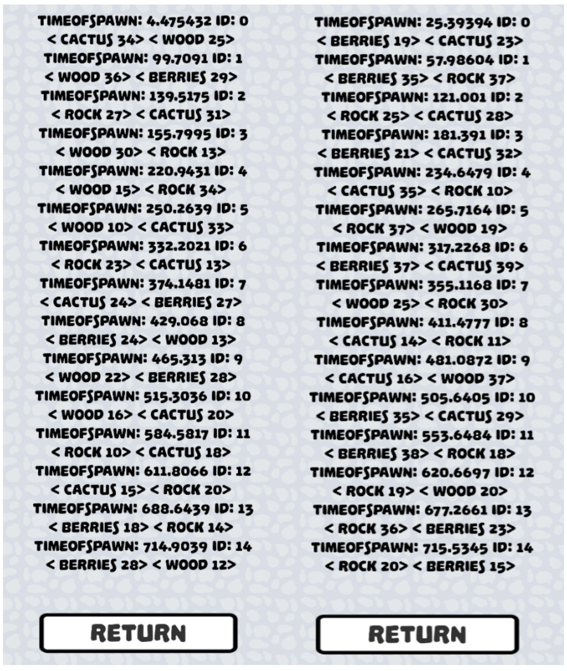
the NPC reaches the first point he moves on to the second point, when he reaches the second point he moves to the third point, and so forth. when the NPC reaches the last point he loops back to the first point in a perpetual cycle.

Using this simple and mechanistic behavior for mapping the movement of NPCs allowed us to run scenarios with more that 100 NPCs and give us a better idea of how the model iterations were behaving in those scenarios.

It's important to note that all the NPCs on all the scenarios are running at the same time and not only the ones presented near the player camera. There are multiple techniques for allowing hundreds of NPCs to move using some sort of path finding with collision avoidance, but because of time constraints the decision was made to make NPCs more “simple”.

There are many ways to allow for hundreds (if not thousands) of NPCs to coexist on a single world environment. One of the ways could be to add a Level of Detail to the behavior of NPCs. A possible inclusion of levels of detail will be talked about further ahead on the future work section.

4.5.3 Message Injection


```
TIMEOFSPAWN: 4.475432 ID: 0
< CACTUS 34> < WOOD 25>
TIMEOFSPAWN: 99.7091 ID: 1
< WOOD 36> < BERRIES 29>
TIMEOFSPAWN: 139.5175 ID: 2
< ROCK 27> < CACTUS 31>
TIMEOFSPAWN: 155.7995 ID: 3
< WOOD 30> < ROCK 13>
TIMEOFSPAWN: 220.9431 ID: 4
< WOOD 15> < ROCK 34>
TIMEOFSPAWN: 250.2639 ID: 5
< WOOD 10> < CACTUS 33>
TIMEOFSPAWN: 332.2021 ID: 6
< ROCK 23> < CACTUS 13>
TIMEOFSPAWN: 374.1481 ID: 7
< CACTUS 24> < BERRIES 27>
TIMEOFSPAWN: 429.068 ID: 8
< BERRIES 24> < WOOD 13>
TIMEOFSPAWN: 465.313 ID: 9
< WOOD 22> < BERRIES 28>
TIMEOFSPAWN: 515.3036 ID: 10
< WOOD 16> < CACTUS 20>
TIMEOFSPAWN: 584.5817 ID: 11
< ROCK 10> < CACTUS 18>
TIMEOFSPAWN: 611.8066 ID: 12
< CACTUS 15> < ROCK 20>
TIMEOFSPAWN: 688.6439 ID: 13
< BERRIES 18> < ROCK 14>
TIMEOFSPAWN: 714.9039 ID: 14
< BERRIES 28> < WOOD 12>
```

```
TIMEOFSPAWN: 25.39394 ID: 0
< BERRIES 19> < CACTUS 23>
TIMEOFSPAWN: 57.98604 ID: 1
< BERRIES 35> < ROCK 37>
TIMEOFSPAWN: 121.001 ID: 2
< ROCK 25> < CACTUS 28>
TIMEOFSPAWN: 181.391 ID: 3
< BERRIES 21> < CACTUS 32>
TIMEOFSPAWN: 234.6479 ID: 4
< CACTUS 35> < ROCK 10>
TIMEOFSPAWN: 265.7164 ID: 5
< ROCK 37> < WOOD 19>
TIMEOFSPAWN: 317.2268 ID: 6
< BERRIES 37> < CACTUS 39>
TIMEOFSPAWN: 355.1168 ID: 7
< WOOD 25> < ROCK 30>
TIMEOFSPAWN: 411.4777 ID: 8
< CACTUS 14> < ROCK 11>
TIMEOFSPAWN: 481.0872 ID: 9
< CACTUS 16> < WOOD 37>
TIMEOFSPAWN: 505.6405 ID: 10
< BERRIES 35> < CACTUS 29>
TIMEOFSPAWN: 553.6484 ID: 11
< BERRIES 38> < ROCK 18>
TIMEOFSPAWN: 620.6697 ID: 12
< ROCK 19> < WOOD 20>
TIMEOFSPAWN: 677.2661 ID: 13
< ROCK 36> < BERRIES 23>
TIMEOFSPAWN: 715.5345 ID: 14
< ROCK 20> < BERRIES 15>
```

Figure 4.4: In this example we can see the different types of message sequences being procedurally generated.

The first versions of the model were developed very early, and because of this we could see very early how the model was behaving on multiple NPCs on early scenarios. This was very useful if we wanted to move the player to a certain place, perform an action on an entity and see how that message propagated in the system.

But there were different types of tests we needed to make. One of those tests was to inject multiple messages at specific points in time on a random NPC. Every scenario was created with a message sequence and it was possible to repeat a single scenario as many times as needed ant try to see patterns emerging. It was possible to shuffle message sequences, determine what types of messages were being procedurally generated and the interval between each message (see Figure 4.4).

4.5.4 Event Spawner

Message sequences were not the only way to test messages in the different simulations. Besides the actions done by the player, we could also want to make an action happen when an NPC passed through a certain part of the map. To make this happen a new type of game entity was created called the *Event Spawner*.

The *Event Spawner* worked like a trap. When an NPC passed on top of the spawner, the spawner would send the message he was holding to that specific NPC. This was an important tool in initializing certain NPCs with certain messages, as well as testing how the model prioritizes old/new messages.

4.6 Map Loading and Scenario Tests

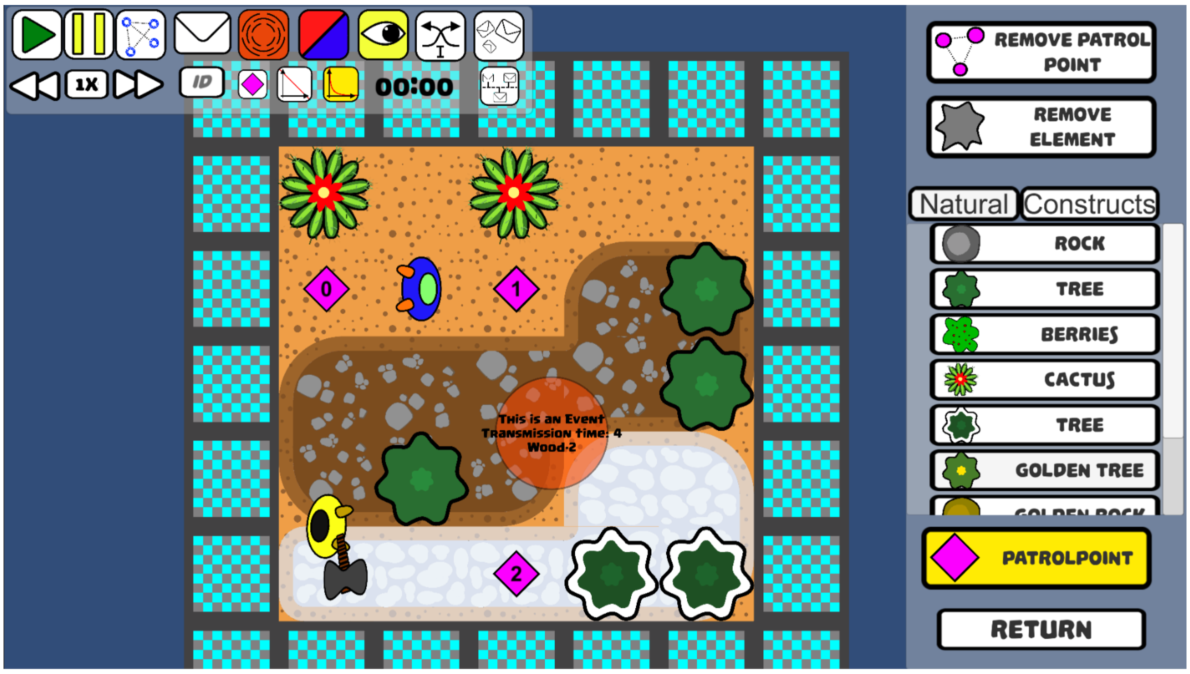


Figure 4.5: Example of a very simple Map where all map components are being saved.

The terrain maker was the most important tool to save time making different scenarios, but the most important tool that allowed us to test the model was the ability to save entire scenarios and load them afterwards. There were multiple components needed to save a *Scenario* and when the game was loaded it was important that all this components were present. That way there was no time lost in “setting up the experiment” when testing the effect of a variable on an outcome.

On Figure 4.5 we can see the different components we are able to save and load in real time when we want to test a specific scenario and in Table **??** we can see how the map data is being saved.

	Data
Message Id Count	1
Foreground Layer	-1 -1 -1 -1 -1 -1 -1;-1 -1 -1 4 4 8 -1;-1 -1 -1 4 4 8 -1; -1 -1 -1 4 4 8 -1;-1 -1 4 4 8 8 -1;-1 -1 4 4 8 8 -1;-1 -1 -1 -1 -1 -1 -1
Background layer	-1 -1 -1 -1 -1 -1 -1;-1 5 5 5 5 5 -1;-1 5 5 5 5 5 -1; -1 5 5 5 5 5 -1;-1 5 5 5 5 5 -1;-1 5 5 5 5 5 -1;-1 -1 -1 -1 -1 -1 -1
entities	n 4 2 -2;n 4 1 -2;n 1 2 0;n 1 2 1;n 1 -1 -1;n 3 -2 2;n 3 0 2
Patrol Points	-2 1#@0 1#@0 -2#
NPCs	-1 1;Bob;0.1254902,0.09411765,0.9764706,0.5176471,1,0.427451,1,0.4509804,0.03137255;Rock 0.2857143, Wood 0.1428571,Cactus 0.5714286;;0,1,2;0.5;0.5;0#
Event Spawner	0.3706408 -0.5188599 0.6 4#This is an Event#Wood 2
MessageSequence	34.21652#0&4&This is an event Cactus and Berries&Cactus 64,Berries 38@ 80.7889#1&4&This is an event Wood and Cactus&Wood 26,Cactus 61@ 108.7895#2&4&This is an event Wood and Berries&Wood 56,Berries 76@ 164.9493#3&4&This is an event Rock and Cactus&Rock 73,Cactus 55@ 230.0608#4&4&This is an event Cactus and Rock&Cactus 27,Rock 72@ 260.1839#5&4&This is an event Rock and Berries&Rock 53,Berries 71@ 346.4899#6&4&This is an event Rock and Cactus&Rock 49,Cactus 56@ 379.2432#7&4&This is an event Cactus and Berries&Cactus 72,Berries 22@ 440.3197#8&4&This is an event Rock and Cactus&Rock 69,Cactus 67@ 476.7178#9&4&This is an event Wood and Rock&Wood 73,Rock 70@ 541.46#10&4&This is an event Berries and Rock&Berries 71,Rock 78@ 588.2433#11&4&This is an event Wood and Berries&Wood 46,Berries 25@ 616.9537#12&4&This is an event Wood and Berries&Wood 37,Berries 35@ 693.5572#13&4&This is an event Wood and Cactus&Wood 59,Cactus 55@ 727.7506#14&4&This is an event Berries and Rock&Berries 40,Rock 54

Table 4.1: Representation of the data being saved by our map maker.

On the next sections we will explain how the different maps parts are being saved and why they are needed.

4.6.1 Message Id Count

Every message has a unique ID and the game must keep track of all the events that have already happened. If $MessageIdCount$ was not saved, the next time an event happened in the world he could end up with an ID that could already exist on a message that some other NPCs already had. Since two messages are the same if they have the same ID,if $MessageIdCount$ was not saved a situation might happen where different events in the game became the same one.

4.6.2 Foreground Layer and Background layer

This values are stored column by column and separated by a comma. In this example we have a 7x7 matrix of values where $-1 = EmptyCell$, $4 = Dirt$, $8 = Snow$ and $5 = Sand$

Each type of terrain is represented with an unique ID. If the designer wanted to add a new type of terrain, he would simply need to draw the textures and give that terrain a unique ID.

Our terrain in the above example would look like this:

-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	4	4	-1
-1	4	4	4	4	4	-1
-1	4	4	4	8	8	-1
-1	8	8	8	8	8	-1
-1	-1	-1	-1	-1	-1	-1

Table 4.2: Example of how the terrain is saved on the foreground layer

4.6.3 Entities

Represent all the objects placed on the map (like trees, rocks, fences, buildings). Each entity can be represented by:

$\{<Type><Number><PositionX><PositionY>\}$

Where $Type$ represents what type of object it is (example: Natural or Construct), $Number$ represents the number of that object on the list and $PositionX/PositionY$ represent where that object should be placed.

Different entities are separated by a comma. In our example, when we say 'n 4 2 -2' we are saying that the 4th object on the list of 'natural' objects should be placed on the position (x=2, y=-2). On this example, this object corresponds to the snow tree on the bottom right corner of the map.

4.6.4 Patrol Points

Patrol points are separated from each other by a '@' symbol. Each patrol point is represented by:

$\{<PositionX><PositionY>\}$

4.6.5 NPCs

Each NPC is separated by the symbol '@' and is represented by:

$\{<PositionX> \quad <PositionY>;<Name>;<NpcColors>;<NpcInterests>;<NpcAquaintances>;<NpcPatrolPoints>;<AssertivenessLevel>;<CooperativenessLevel>;<NpcType>\#<Messages>\}$

$NpcColors$ are represented by:

$\{<BodyColor>,<HeadColor>,<HandsColor>\}$

Each $Color$ is represented by:

$\{<R> \quad <G> \quad <B>\}$

Each $NpcInterest$ is separated by a comma and is represented by:

$\{< InterestName > \quad < InterestWeight >\}$

Each $NpcAquaintance$ is separated by a comma and is represented by:

$\{< FriendName > \quad < FriendshipLevel >\}$

$NpcPatrolPoints$ are identified by their id and each of them is only a number separated by a space.

$AssertivenessLevel$ and $CooperativenessLevel$ represent the respective values of assertiveness and cooperativeness of the NPC we are trying to load.

NPCs come in different shapes and sizes, and can be made to have different kinds of behavior. The variable $NpcType$ represents what kind of NPC is going to be loaded. There are two different types of NPCs in out prototype: Guardians and Villagers.

4.6.6 Event Spawner

Event Spawner are able to give a specific message to a player when the player collides with it. As we can see in Figure 4.5, the event is represented by the red circle. When some NPC collides with it he receives the message it contains, removing the spawner after the message is received by the NPC.

This spawner were a tool used to test the propagation of different messages and no longer need to be saved on the map file if a finished product around this tool (like a ”Game Maker”) was to be made.

Spawner are separated by '@' and each spawner can be represented by:

$\{< PositionX > \quad < PositionY > \quad < MessageTransmissionDistance > \quad < MessageTransmissionTime >$
$\# < MessageDescription > \# < MessageTags >\}$

Where $MessageTransmissionDistance$ describes the distance an NPC must be to receive the message from the spawner.

4.6.7 Message Sequence

$MessageSequence$ refers to the messages that are given to random NPCs at specific times. They are procedurally generated and separated by @. Each message has the following structure:

$\{< MessageId > \quad < MessageTransmissionTime > \& < MessageDescription > \& < MessageTags >$
$\}$

Each $MessageTag$ is separated by a comma and composed of:

$\{< TagName > \quad < TagWeight >\}$

4.7 NPC Memory and Code Optimization

When we talked before about the necessity of limiting the memory of NPCs (how many messages they can retain before they are forced to discard old messages) we didn't go into detain about how much memory we would allow an NPC to have.

The most important way to know how many NPCs are going to talk about a specific message is through the message weight. If a player cuts down 10 trees a certain NPC could end up with 10 messages saying "The player cut down a tree". It it really necessary to store this many repeated messages? Probably not. Important messages are often singular events like "The king is dead" or "The player killed someone" and not necessarily come from the player doing repetitive tasks.

So there is a good reason not to store a lot of data. If the designer chooses to create NPCs that change their perception about the player, they only need to change the state of an NPC when the message is received and should not depend on the current messages the NPC is holding on memory. So, *"the primary objective of a message is to be passed along to other NPCs. There is no point in knowing the same thing multiple times."*

In the case of the player cutting down a tree, with unlimited memory the NPC would be filled with unnecessary information. With limited memory, then a new message about "cutting down a tree" was received, it would replace an older and similar message of "cutting down a tree".

As an example, imagine that every NPC has infinite memory and has accumulated on average 1000 messages over time. For each possible conversation between two NPCs, they would have to go through all of their messages in order to determine what message is the most interesting for them to talk about. This is very computer expensive in later stages of the game where the player has already performed a lot of different actions on the world.

We could say that a message eventually would be forgotten because of the decay, but even on that situation it would take a while. On the final working prototype we gave each NPC a limited memory of 3 messages. On bigger games that this one the designer might want 5, 7 or even higher memory space for each NPC, even going as far as mapping each NPC to have a memory of different size.

4.8 Simulation Outputs

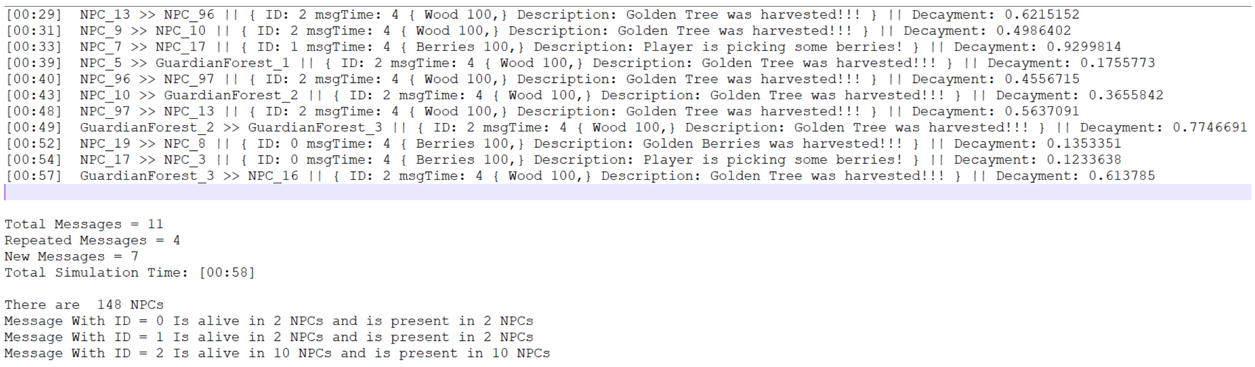
Data outputs allowed us to get a good idea about what was happening during a simulation. With fixed simulation times of 20 minutes, 40 minutes and even 1 hour, we could draw graphs, see the distribution of messages across the world and count how many messages got lost forever, came back to life and how many different NPCs did it reach.

There were three two main simulation outputs used for determining the general "health" of our model:

1. Statistical data through text outputs.
2. Graphs that show how many messages were being exchanged every X minutes or every X seconds on specific locations of the map.

4.8.1 Statistical Data

The first data outputs registered on the simulations was done through text. We logged not only what messages where being exchanged and at what time, but also the ratio of repeated/new messages and how many messages were able to reach different NPCs (as we can see in Figure 4.6)



```
[00:29]  NPC_13 >> NPC_96 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.6215152
[00:31]  NPC_9 >> NPC_10 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.4986402
[00:33]  NPC_7 >> NPC_17 || { ID: 1 msgTime: 4 { Berries 100,} Description: Player is picking some berries! } || Decayment: 0.9299814
[00:39]  NPC_5 >> GuardianForest_1 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.1755773
[00:40]  NPC_96 >> NPC_97 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.4556715
[00:43]  NPC_10 >> GuardianForest_2 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.3655842
[00:48]  NPC_97 >> NPC_13 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.5637091
[00:49]  GuardianForest_2 >> GuardianForest_3 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.7746691
[00:52]  NPC_19 >> NPC_8 || { ID: 0 msgTime: 4 { Berries 100,} Description: Golden Berries was harvested!!! } || Decayment: 0.1353351
[00:54]  NPC_17 >> NPC_3 || { ID: 0 msgTime: 4 { Berries 100,} Description: Player is picking some berries! } || Decayment: 0.1233638
[00:57]  GuardianForest_3 >> NPC_16 || { ID: 2 msgTime: 4 { Wood 100,} Description: Golden Tree was harvested!!! } || Decayment: 0.613785


Total Messages = 11
Repeated Messages = 4
New Messages = 7
Total Simulation Time: [00:58]

There are  148 NPCs
Message With ID = 0 Is alive in 2 NPCs and is present in 2 NPCs
Message With ID = 1 Is alive in 2 NPCs and is present in 2 NPCs
Message With ID = 2 Is alive in 10 NPCs and is present in 10 NPCs
```

Figure 4.6: Example of text output data.

4.8.2 Data Graphs for Analyzing Message Propagation

Data graphs were created a lot further ahead in the development of the model, and by this stage we already had an almost complete version of the final map. Because of this we could register messages being propagated on different parts of the map (See Figure 4.7) instead of the entire map.

We could also get important data like the graph of messages that were alive(see Figure 4.8) or how many exist in the world (even when they have maximum decay). Besides the compact graph version, we also had as output a graph for each zone if we wanted to go into a particular zone of the map.

We could observe how a message that got started on $Location_A$, and how it propagated to reach $Location_B$. It allowed us to see that messages are created different and that the NPC that gets $Message_A$ at a certain point of the game is the most important factor in determining if $Message_A$ is going to spread or going to die with him.

When we talk about a message and say that it is “alive” what we are saying is that the decay of this message has not reached the max value yet and because of this be considered by an NPC when choosing what message he is going to talk about.

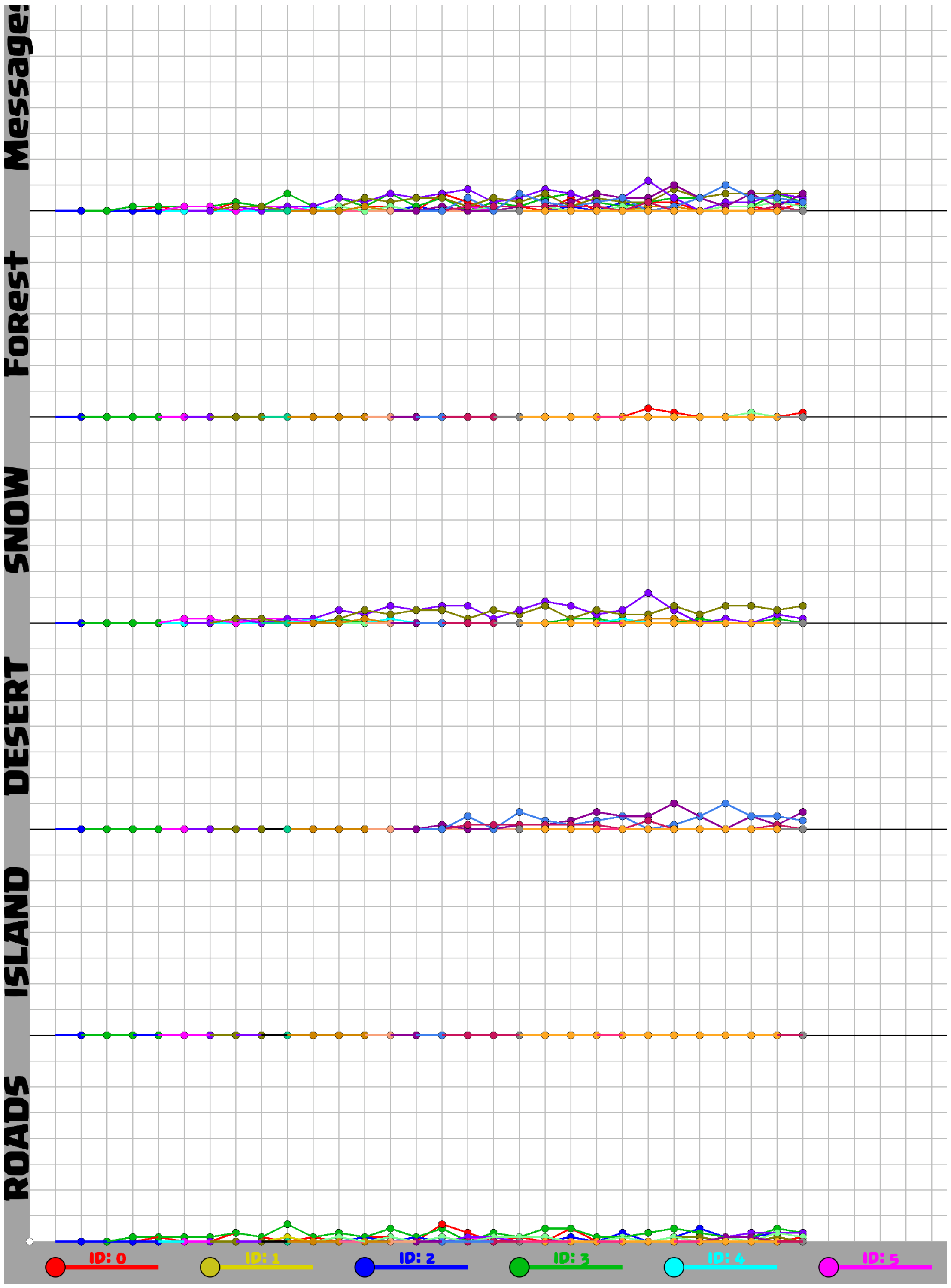


Figure 4.7: Here we can see how messages propagate over time in different parts of the map. The X-Axis represents the time in minutes and the Y-Axis represents the number of messages exchanged between NPCs during that minute.

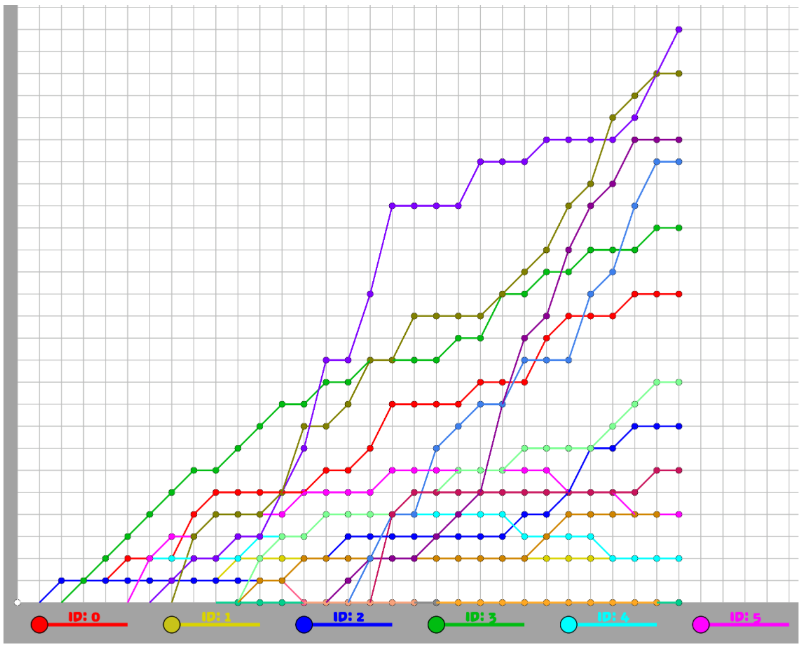


Figure 4.8: Here we can see, for each message, if it is present and alive in the memory of NPCs. The X-Axis represents the time in minutes and the Y-Axis represents the number of messages exchanged between NPCs during that minute.

4.8.2.A Message Assimilation Over Time

One of the important things the graphs in Figure 4.7 and Figure 4.8 were able to detect was how easy or difficult it was for new messages to spread when NPCs already had reached the limit of their memory. When the game starts, since NPCs have nothing in memory and there are no different options to choose from, there is a pike in the popularity of a message because it is the only thing they talk about.

For the model to work, it should be expected that new messages being generated over time behaved in similar ways. The observed behavior of a new message when most NPCs already are at full memory can be described in this three stages:

1. New message is received by one or more NPCs

2. Since the message is new it spreads fast across the immediate circle of connections of those NPCs

3. The rate at which that message is being talk about diminished, and eventually gets forgotten by all NPCs.

4.9 Special Considerations

There were many things that were considered before implementing the final model. As a result of extensive testing during development, some decisions had to be made and some parts of the model had to be adapted. The special considerations described here should be very helpful to any person that tries to implement this model. This considerations are not strict rules, but recommendations and guidelines on right and wrong practices, as well as personal feedback on certain aspects of the model that didn't have the feedback desired.

4.9.1 Predictability of Encounters

Since the designer doesn't know where NPCs are going to be at an exact time (because they could, for example, have stopped to talk to other NPCs) he can't know how a specific message he wants to test is going to spread across the network.

Even with the unpredictable nature of the NPC's state in the world (where are they going to be at a certain time), it was possible to determine that the rate at which messages were being exchanged remained very similar in multiple tests using the same message sequence.

4.9.2 Expected vs Unexpected Behaviors

Even when running the same scenarios over and over again, different results can sometimes be obtained. This happens because our prototype calculates the distance between NPCs every frame and if 3 NPCs are at a distance between themselves that allows for communication to happen, there is no way to know who talks to who. Is a certain situation NPC_A could be talking first to NPC_B and in another one he could be talking first to NPC_C

4.9.3 NPC Interaction and Proximity of Events

The distance required for two NPCs to start talking can change a lot and will end up being a design decision that has a big impact on the game when applying our model. The further away the designer allows two NPCs to communicate with one another, the higher is becomes the chance of a message being communicated between them, since they can talk over long distances.

The distance by which NPCs can see events happening and receive messages from those events is very important to determine not only if an NPC catches a message of an event but also how many NPCs get that message. If all NPCs in the map see an event, to them that event is considered old, since all of them have it in memory.

It is very rare for a message to be known to every NPC in the world, so most NPCs will trade messages that are new to them instead of messages that are old to them (meaning they have heard of those messages before).

In this model NPCs will always prioritize new messages over old messages (assuming all other values are the same).

4.9.4 Conversation Duration

The duration of a conversation not only affects how much time it takes for an NPC to transmit a message to another NPC, but also how the values of $Assertiveness$ and $Cooperativeness$ increase and decrease over time.

This is one of those variables that only through testing and manipulation can a designer make sure that the rate of transmission of messages is happening the way he wants.

Since NPCs cant receive messages from other NPCs not involved on the conversation, if messages take too long to transmit there will be less messages circulating on the network and as a result the player might have to wait for a certain message to reach the NPC he wants.

If conversations are too fast, the rate at which $MessageA$ reaches $LocationA$ will depend primarily on the movement speed of the NPC who has that message and to what destination is he moving to.

If messages take no time to transmit, $NPCs$ will not have time to decrease their values of $Assertiveness$ and $Cooperativeness$. Because of this, they will keep talking forever, never moving from their position or changing their behavior. One way to fix this is to lower the values of $Assertiveness$ and $Cooperativeness$ instantly after each conversation, instead of decreasing this values over time.

4.9.5 New Messages vs Old Messages

Old messages behave slightly differently from new messages. New messages are interpreted as old messages that don't have decay. Imagine a scenario where an NPC receives two messages. He first receives $message_A$ and after a couple of seconds receives $Message_B$. Even though both messages are received with a few seconds apart, the fact that the first message is new and has $Decay_A = 1$ and the second message slightly older but with a fast decrease in decay value to a point where $Decay_B = 0.3683$, allows for the algorithm to prioritize a lot more messages that are new instead of messages that are older.

If we had a linear decay on messages however, instead of having $message_A$ and $message_B$ with a big difference in $Decay$ we would end up with similar decays like $Decay_A = 1$ and $Decay_B = 0.95$. At the moment the NPC is receiving $message_B$, since both new and old messages are so similar in their values of decay, the distinction between Old/New messages is barely noticeable, if both messages have similar *TAGs* and *Weights*

During the development of the model I tried working with values of $Decay$ changing in a linear way and in an exponential way. Exponential decrease in $Decay$ produced a lot better results since NPCs had a better chance of gathering new messages and forgetting old ones. Bellow is the Formula 4.1, used to change the decay value in an exponential way, first dropping faster and then dropping lower.

$$Decay = Decay - Decay * DecayRate * \Delta t \tag{4.1}$$

When a message is created, it has an initial decay of $Decay = 1$ and reaches max decay at $Decay = 0$. The $DecayRate$ determines how fast the decay is going to reach $Decay = 0$ and Δt represents the time between each game frame.

The next frame update of the $Decay$ value is dependent on the current $Decay$. If the decay starts at $Decay = 1$ the value will drop a lot faster at first and slower later on. The bigger the variable $DecayRate$ is, the faster the $Decay$ value is going to drop.

Although not considered in this dissertation, different types of messages can have different types of decay. This is something to be explored in future works.

4.9.6 Community Formation

In the first game prototype we created one town, where the majority of NPCs lived, and a few other NPCs scattered around the four corners of the map (see Figure 4.9). This first prototype was important to measure how a message can spread among a community of NPCs with different types of interests. We were also able to record what type of NPC captures what type of message.

In the final prototype there are 4 major Communities. Because it was not the focus of this dissertation no allow for communities to form naturally in the world, on the final prototypes some towns were created and NPCs were made to behave like they were in a big community. This allowed us to see how difficult it is to pass information across communities and how more isolated communities have a harder time in knowing about events happening near other communities.

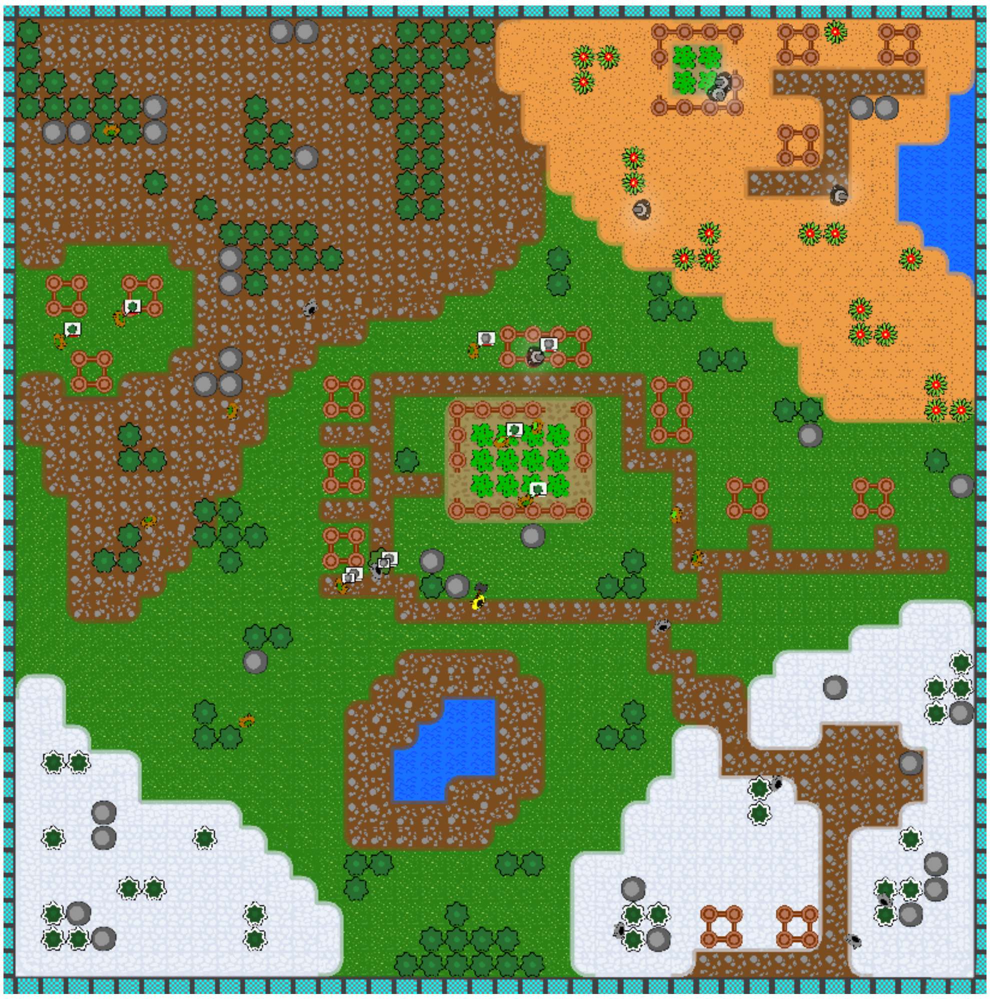


Figure 4.9: In this image we can see the a big community of NPCs in the middle and fewer NPCs dispersed across the map. Some NPC's moved from the periphery of the map to the town center.

5

Evaluation

Contents

To test our model, two different versions of the prototype were elaborated:

1. **Version A:** This version did not make use of our *NPC Communication Model* for sharing messages. Instead it uses an alternate version of our model where messages are transmitted to all NPCs in the map instantly. This behavior was used in order to simulate the behavior of games that typically use a centralized system for how information is gathered and shared among NPCs.
2. **Version B:** We used our model to shape the behavior of NPCs. This time, instead of the player's actions being sent as messages instantly to all NPCs, those messages propagate over time in a network of NPCs, each with its own interests.

The first version did not use our model and as such, every time the player grabbed a relic on the map, a message would be sent to every NPC that was a guardian. Guardians would from that moment on stay on high alert, chasing the player if the player was at a close proximity to a guardian.

On the second version of the prototype we used our model to shape the behavior of NPCs. This time, messages about the player's actions would only be known to NPCs that were close to the player. In this version NPCs would spread messages about what the player was doing over time. If a message relating to the player catching a relic reached a guardian, that specific guardian would stay on alert, going after the player if he saw the player.

5.1 Data collection methods

There were two different data collection methods used to reach a conclusion regarding how viable a model like this one would be useful in video games.

The first method was through the use of a simple questionnaire with the feedback from the player regarding the play through of each version of the prototype. Using a questionnaire, we wanted to observe if our model increased the gaming experience of the player and if the player was able to understand how NPCs were communicating between themselves.

The second type of data collected was in the form of a database, where we were recording what the player was doing in different parts of the game. This type of data was important in having a deeper look at how the player played the game in each version. We will describe in more detail each data value captured in the section bellow.

5.1.1 Questionnaire Data

For this evaluation, we asked participants to play both versions of the prototype, with players randomly choosing what version they wanted to play first. At the end of playing each version each player was asked two questions (The original questionnaire is in Appendix A):

1. Rate his overall gaming experience in a scale of 1 to 7.
2. How credible was the communication between NPCs, meaning how easy it was for the player to understand what type of things NPCs were talking about. This was also a rating on a scale of 1 to 7.

If the players gave different ratings on these 2 questions in each version, we would then ask why they rated them differently. This feedback was important in knowing if our model worked correctly and if not what could be done to further make it better in the future.

5.1.2 Game Data Collection

Each time a player tried one of the versions, a few values were being sent to a database. This values were:

1. **Relics Grabbed:** We not only knew what relic was grabbed by the player but at what time was that relic grabbed. This was useful in seeing if the player was getting better at the game and, with increased difficulty, if the player took more time in grabbing the next relic.
2. **Grabbed by Guardians:** We also wanted to see how many times the player has been grabbed by guardians and how he adapted to the game, managing to escape being chased by NPCs. If the player was dying a lot in later stages it could mean that the game was being too difficult and if he died a lot more at the start it could mean that the player was having difficulties with the way he controlled the player.
3. **Tutorial Time:** We wanted also to see how much time each player took to finish the tutorial. This was a good indication of how at ease the player was with the controls and also show how experienced the player was at playing this type of games.
4. **Game Time:** With this data we could see how much time the player took to finish each version of the game and see if the model made it easier for the player to find all game objects or if there was no substantial difference in the time it took to finish the game.

5.2 Player Testing

In this section we describe how we conducted our tests and what were most important measures used in determining if our model worked or not.

We hypothesize that the overall game experience of the player will be significantly better on the version with the model as a result of the communication happening between NPCs. The question relating to the credibility of the communication happening in the world should also be significantly better on the game version with our model present. The ability of the player to complete the objective should not be different on both versions.

The version of the game with our model should also lead to a decrease in objective completion time since we assume that with our model the communication happening in the world would be more believable and play an essential role in the conclusion of the objective.

To compare the different versions of our prototype we used a Wilcoxon signed-rank Test to test game experience and NPC communication with our participants. Both variables tested were ordinal variables on a 7-point Likert scale, where we tested both pairs of questions with both prototypes.

5.2.1 Sample

This evaluation was done with 15 people, with ages ranging from 9 to 32 years old (M = 22.33, SD = 6.332). On the pie charts of Figure 5.1 we can see the different types of users that tested our prototypes.

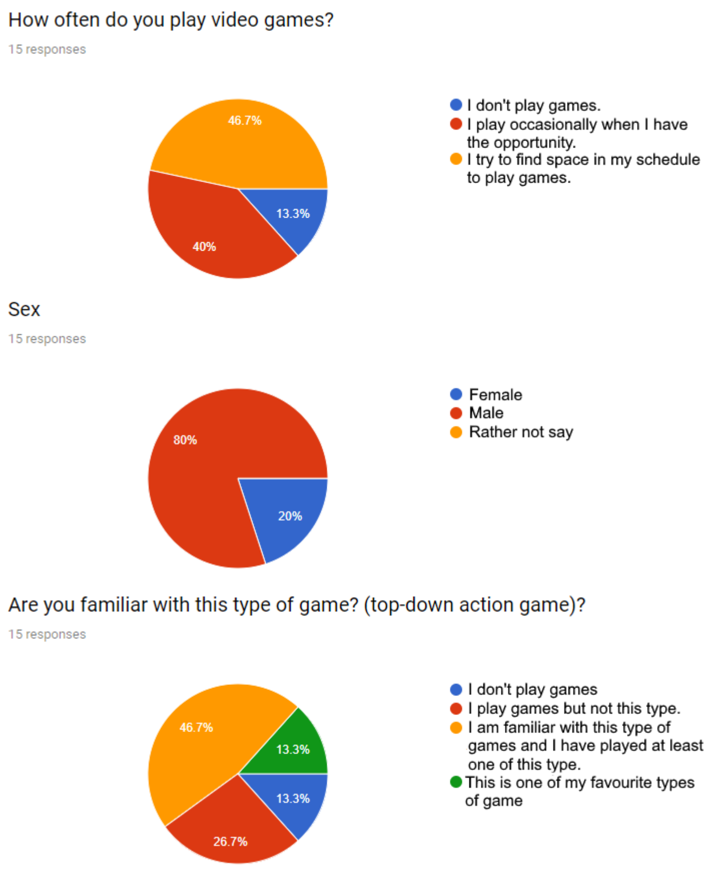


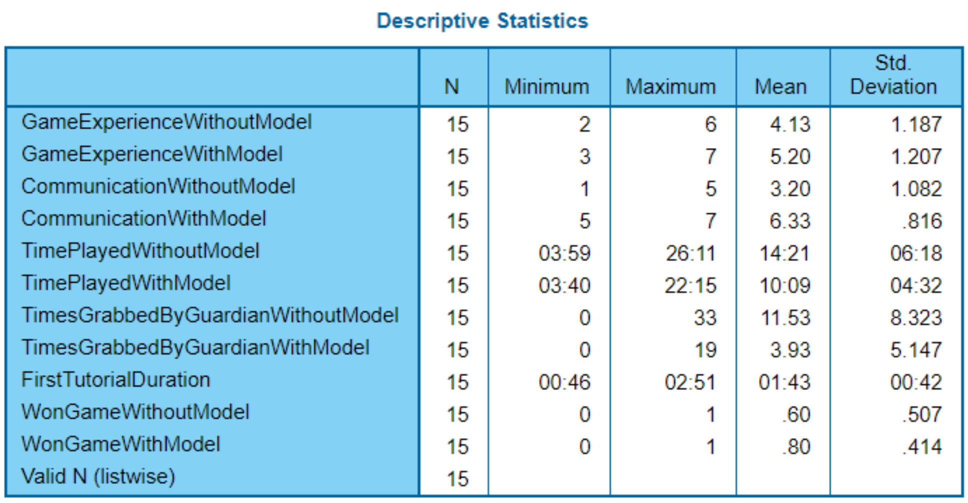
Figure 5.1: In this figure we can observe that 86.7% of participants had experience in playing video games and that 60% of them already played at least 1 game similar to the one on the prototype.

5.2.2 Procedure

Some of the tests were done face-to-face with our participants and other ones were done remotely. All participants were asked to perform the following tasks:

First each player was asked to respond to 4 demographic questions: age, sex, gameplay frequency and familiarity with the game genre. After responding to this demographic questions, each participant was asked to play $Version_A$ or $Version_B$ at random. At the end of playing each version, the participant would rate his overall gaming experience and how well did he understood the communication between different NPCs. They were allowed after playing both version to change the ratings they gave to both questions.They were also asked the reasons for the difference in the given ratings.

5.3 Results



Descriptive Statistics

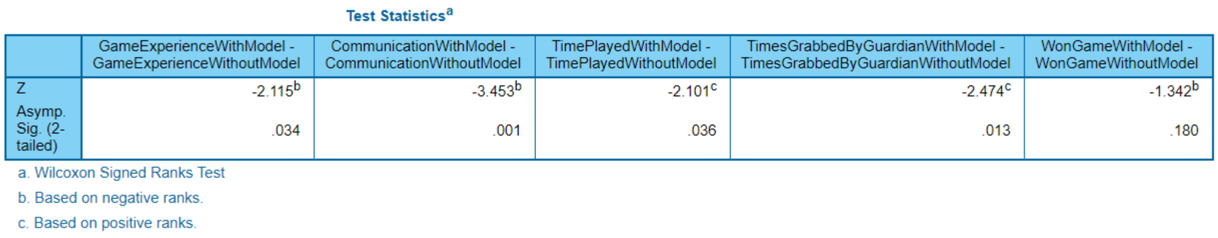
	N	Minimum	Maximum	Mean	Std. Deviation
GameExperienceWithoutModel	15	2	6	4.13	1.187
GameExperienceWithModel	15	3	7	5.20	1.207
CommunicationWithoutModel	15	1	5	3.20	1.082
CommunicationWithModel	15	5	7	6.33	.816
TimePlayedWithoutModel	15	03:59	26:11	14:21	06:18
TimePlayedWithModel	15	03:40	22:15	10:09	04:32
TimesGrabbedByGuardianWithoutModel	15	0	33	11.53	8.323
TimesGrabbedByGuardianWithModel	15	0	19	3.93	5.147
FirstTutorialDuration	15	00:46	02:51	01:43	00:42
WonGameWithoutModel	15	0	1	.60	.507
WonGameWithModel	15	0	1	.80	.414
Valid N (listwise)	15				

Figure 5.2: Descriptive Statistics of data collected during participant's gameplay of both versions of the prototype where the time is measured in $< minutes >:< seconds >$

As we can see in Figure 5.2, the mean value of the gameplay experience of the players using both versions was $Version_A = 4.13$ and $Version_B = 5.2$. A Wilcoxon signed-rank test showed a statistical significant difference regarding the overall gaming experience when using both models ($Z = -2.115$, $p < 0.05$). This leads us to conclude that a version using our model resulted in an increase in overall gameplay experience of the player, as we expected to happen.

When we talk about the ability of the player to understand how NPCs communicate in the world, there was a significant difference in perception from the players since we have mean values of $Version_A = 3.2$ and $Version_B = 6.33$. A Wilcoxon signed-rank test showed that the use of our NPC communication Model had a statistically significant impact on the way players understood the communication happening between NPCs ($Z = -3.453$, $p < 0.01$) This suggests that the communication between NPCs using our model is a much more desirable feature than a version without it.

We can observe here that the average play time using our model was 10 minutes and 9 seconds in comparison to the 14 minutes and 21 seconds taken to complete the version without our model. This suggests to us that our model helped the player complete his objective faster without compromising his overall gaming experience.



Test Statistics[a]

	GameExperienceWithModel - GameExperienceWithoutModel	CommunicationWithModel - CommunicationWithoutModel	TimePlayedWithModel - TimePlayedWithoutModel	TimesGrabbedByGuardianWithModel - TimesGrabbedByGuardianWithoutModel	WonGameWithModel - WonGameWithoutModel
Z	-2.115[b]	-3.453[b]	-2.101[c]	-2.474[c]	-1.342[b]
Asymp. Sig. (2-tailed)	.034	.001	.036	.013	.180

a. Wilcoxon Signed Ranks Test

b. Based on negative ranks.

c. Based on positive ranks.

Figure 5.3: Here we summarize the test results, and observe a positive impact of our model on our game.

Another possible reason of a significant difference in gameplay time could also be an increase in difficulty on the version without our model, since all guardians are after the player the moment the player grabs the first relic. Even with an increase in difficulty, the Wilcoxon test showed no statistical significant difference in the number of times players were able to complete the objective ($p = 0.18$).

The Wilcoxon signed-rank test (see Figure 5.3) also shows us values of $p < 0.05$ for the time it took to complete the objective, so there is a statistical significant impact of our model on the prototype in this regard. The player could have taken less time in completing the objective using our model because not all NPCs knew the player's actions and as a result only some of them chased the player. This behavior could have helped the player have a better reading of what was happening in the game and made him more efficient in the conclusion of the objective.

5.3.1 Player Feedback

We also were able to gather of player feedback from our questionnaire. Many players said they were able to see their actions reflected on the NPCs behavior. Some players also noticed that when they gathered a relic, one of the NPCs that was close by moved towards the guardian to report that player's actions. Since that point they avoided grabbing relics when there were NPCs close by that could catch them in the act.

Some players also reported that guardians came after them after they had gathered a relic. Those players clearly understood that they should not grab relics when they were in perception distance of guardians.

One of the participants commented: *On the first version (without the model) I understood what they (the guardians) were talking about but I did not understand the purpose of their conversation. On the* *second version (with our model) I noticed that when I broke a relic one of the NPCs was going to tell a guardian that I had done and after this he started following me. One of the times, when I was coming back for the second time to the top part of the map, the guardians had changed and were now looking for me.*

Another participant told us: *On version B (with our model) the ghosts (guardians) seemed more capable of communicating with other NPCs that on version A (without our model). On this version (Version B) they also seemed to understand what the player was catching and not just that the player wanted to catch*

This kind of feedback was extremely important for us to understand our participants reason for giving different ratings for both versions and understand better the impact our model had on their gameplay experience.

6

Conclusion

With this work we presented a model for how NPCs in a simulated world environment could communicate between themselves in a more believable way. This type of communication would be not only important for a possible change in behavior from the NPCs but for the player to see that the world he was experiencing was more complex and believable. Our model was able to describe the world by different types of parts and explain how those parts fit together to make a different and more believable type of communication happen between NPCs.

We started by researching how current games try to change the behavior of NPCs to make the player more engaged and interested in the world and the story. Like the games described in the Related Work section, our model offers one more tool for increasing the complexity of the game world and the NPCs who inhabit that world, giving the player an overall better gaming experience.

It is also quite important to highlight the incremental development of different types of tools, all of them created at a certain point to test our model and how did he behave under stress. This tools also allowed us to play around with different model parameters and observe how the world reacted to them. Our model was testes over and over again, perfected until we reached a working version described in this document.

Our approach was centered around the player, with the focus being on the gameplay experience a player would have if he was to play a system with our model implemented. Each incremental step we asked ourselves: ”How does this change make the game better?” and ”What different possibilities can emerge if we make this changes?”

We then started creating simple gameplay scenarios and as the different maps became more complex, so did the behavior of our NPCs. From just 20 NPCs to start with we expanded the map until we had a finished prototype with over 100 NPCs, all moving around the map sharing sharing messages between themselves. Statistical data was collected during every stage of development and this data was worked upon until the values we were getting were in line with the ones we wanted.

The elaboration of tests with human players, where they were asked to play two different versions of the game, one using our model and the other using a standard approach to how information is shared in current video games was the final way by which we determined if our model was functional.

We had very positive results from our participants in the final testing phase. Participants were not only able to understand what the NPCs were talking about, but also why they were talking about it. They were able to adapt their gameplay to the communication happening between NPCs and as a result changed their behavior in order to complete the objective faster.The participants in our tests also reported an overall better gaming experience when they were using our model.

We believe that our model is a step forward in making more realistic worlds for the player to explore, increasing the replayability of the game and possibly enabling emergent gameplay situations to occur.

Future Work

There are several ways of continuing the work executed throughout this project, but we are just going to point out the most important ones. Some components that were excluded during the development of the final prototype (for the sake of simplicity and saving time) but could be implemented in further developments of this model.

They are: *Friendship Level*, *Goal Oriented Movement* and *Level of Detail*

This components can not only make the communication between NPCs seem more believable, how they behave in the presence of each other and also how they move and make decisions in the world.

Friendship Level

As discussed in previous sections, the friendship level between NPCs could play an important part in determining who receives what messages. It is not an easy component to balance and can possible break the gameplay experience of the player. Further solutions to this component could be explored as the model is currently able to support the usage of friendship level between NPCs

Goal Oriented Movement

As described above when talking about “Smart Terrain”, it would be very interesting to know how information would be shared if NPCs moved based on their personal goals instead of a fixed pre-established movement. Some locations would naturally become more populated, and some locations could end up without NPCs.

The the communication happening in the world has an effect on NPCs and their goals changed over time, so would their movement change. That way, NPCs that lived very far away could end up living close to one another after some time.

Level Of Detail

A possible future work here would be to make different levels of detail for different types of NPCs. There are many ways of accomplishing this:

1. Disable NPCs that are further away from the player and give more power to NPCs that are closer to the player and on vision of the camera.

2. Give NPCs near the player the power to use path finding techniques and obstacle avoidance and remove power from NPCs that are further away, giving them a simple patrol type of movement.

3. Allow for NPCs that are near the camera to work properly with path finding. NPCs that are further away would only be active based on the importance of their messages.

Option number 1 and option number 2 are very straightforward to understand but option number 3 is not so easy to understand.

As an example for option 3, lets say that Npc_A is very far away from the player. Calculating the distance between the player Npc_A the game reaches the conclusion that only NPCs that have messages with $LevelOfImportance > 4$ would be enabled. Since Npc_A has a message with $LevelOfImportance = 6$, he is able to share that message with disabled NPCs around him. Notice here that even if an NPC is disabled, he must still be able to receive messages from enabled NPCs around him. There is no point in having an enabled NPC far away from the player if all the rest of the NPCs cannot receive that important message.

Bibliography

[1] I. D. Joyce Whang, “Overlapping community detection in massive social networks,” http://bigdata.ices.utexas.edu/project/graph-clustering/.

[2] A.-L. Barabási, *NetworkScience*. Cambridge University Press, New York, NY, 2016.

[3] M.-L. Ryan, *Narrative Across Media: The Languages of Storytelling*. University of Nebraska Press, 2004.

[4] R. J. Trudeau, *Introduction to Graph Theory*. New York: Dover Pub, 1993.

[5] F. K. Wasserman S., *Social Network Analysis: Methods and Applications*. Cambridge University Press, New York, NY, 1994.

[6] A. Alain Barrat, Marc Barthélemy, *Dynamical Processes on Complex Networks*. Cambridge University Press, New York, NY, 2008.

[7] M. Newman, *Networks: An Introduction.* Oxford University Press, 2010.

[8] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, *The Data Mining and Knowledge Discovery Handbook*. Springer-Verlag, New York, USA, 2005.

[9] J. Pan, R. K.; Saramaki, “Path lengths, correlations, and centrality in temporal networks,” *American Physical Society*, July 2011.

[10] A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani, “The architecture of complex weighted networks.” *PNAS*, 2004. [Online]. Available: https://doi.org/10.1073/pnas.0400087101

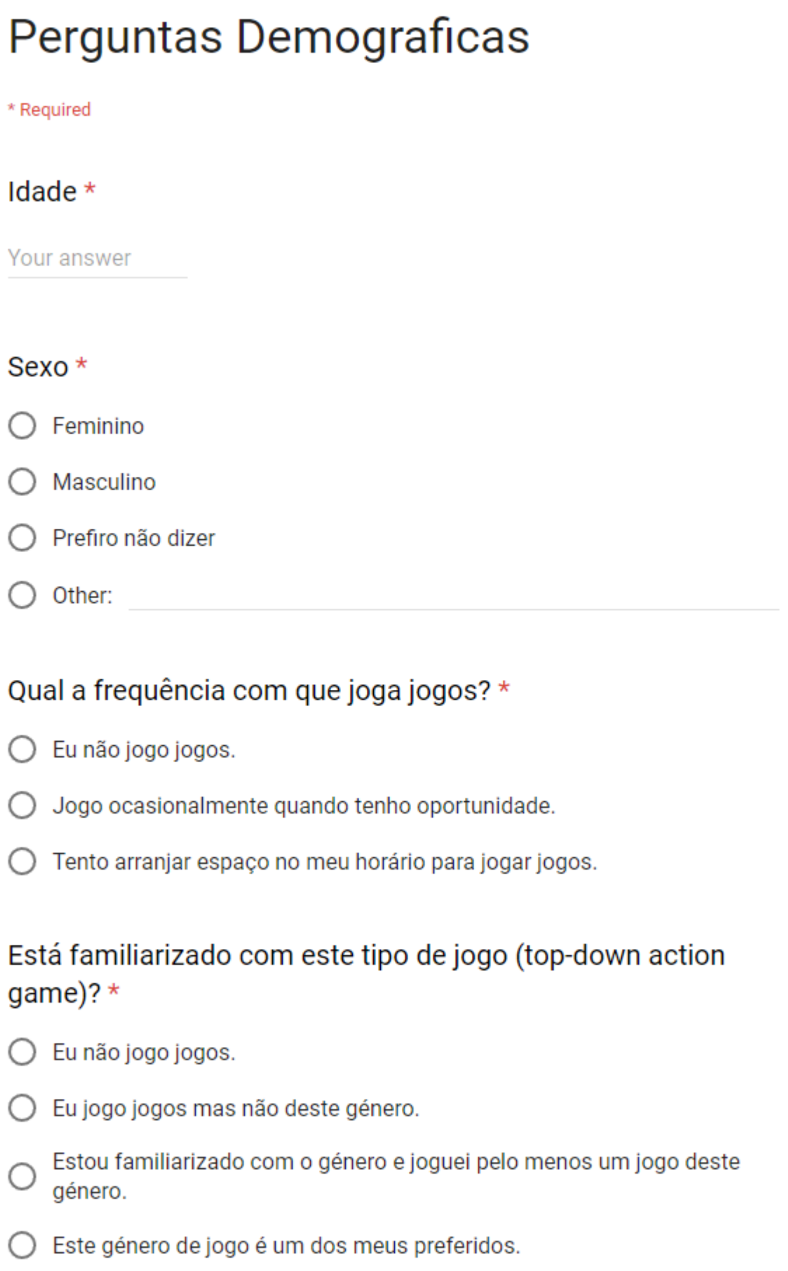
[11] S. Horvath, *Weighted Network Analysis: Applications in Genomics and Systems Biology*. Springer, 2011.

[12] J. Onnela, J. Saramaki, J. Hyvonen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A. Barabási, “Structure and tie strengths in mobile communication networks.” *PNAS*, 2007. [Online]. Available: https://doi.org/10.1073/pnas.0610245104

[13] M. S. Granovetter, “The strength of weak ties,” *American Journal of Sociology*, 1973.

A

Questionnaire and Evaluation Results



Perguntas Demograficas

* Required

Idade *

Your answer

Sexo *

- ◯ Feminino
- ◯ Masculino
- ◯ Prefiro não dizer
- ◯ Other:

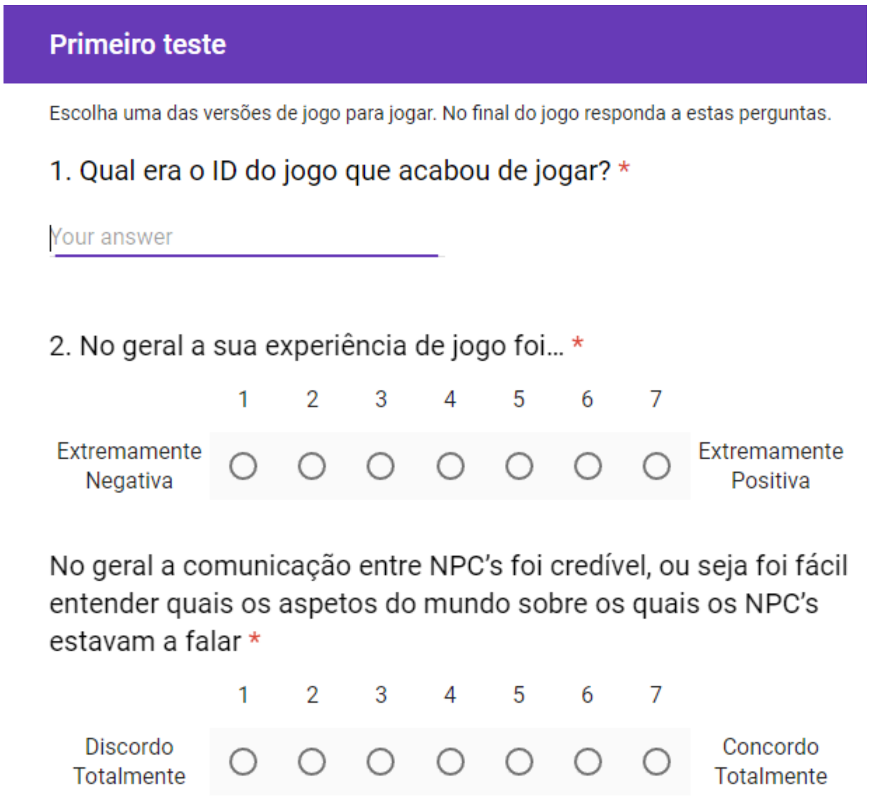
Qual a frequência com que joga jogos? *

- ◯ Eu não jogo jogos.
- ◯ Jogo ocasionalmente quando tenho oportunidade.
- ◯ Tento arranjar espaço no meu horário para jogar jogos.

Está familiarizado com este tipo de jogo (top-down action game)? *

- ◯ Eu não jogo jogos.
- ◯ Eu jogo jogos mas não deste género.
- ◯ Estou familiarizado com o género e joguei pelo menos um jogo deste género.
- ◯ Este género de jogo é um dos meus preferidos.

Figure A.1: Page 1 of the questionnaire. In this page we ask the players their age, sex, gameplay frequency and familiarity with the type of game they are going to play next


Primeiro teste

Escolha uma das versões de jogo para jogar. No final do jogo responda a estas perguntas.

1. Qual era o ID do jogo que acabou de jogar? *

Your answer

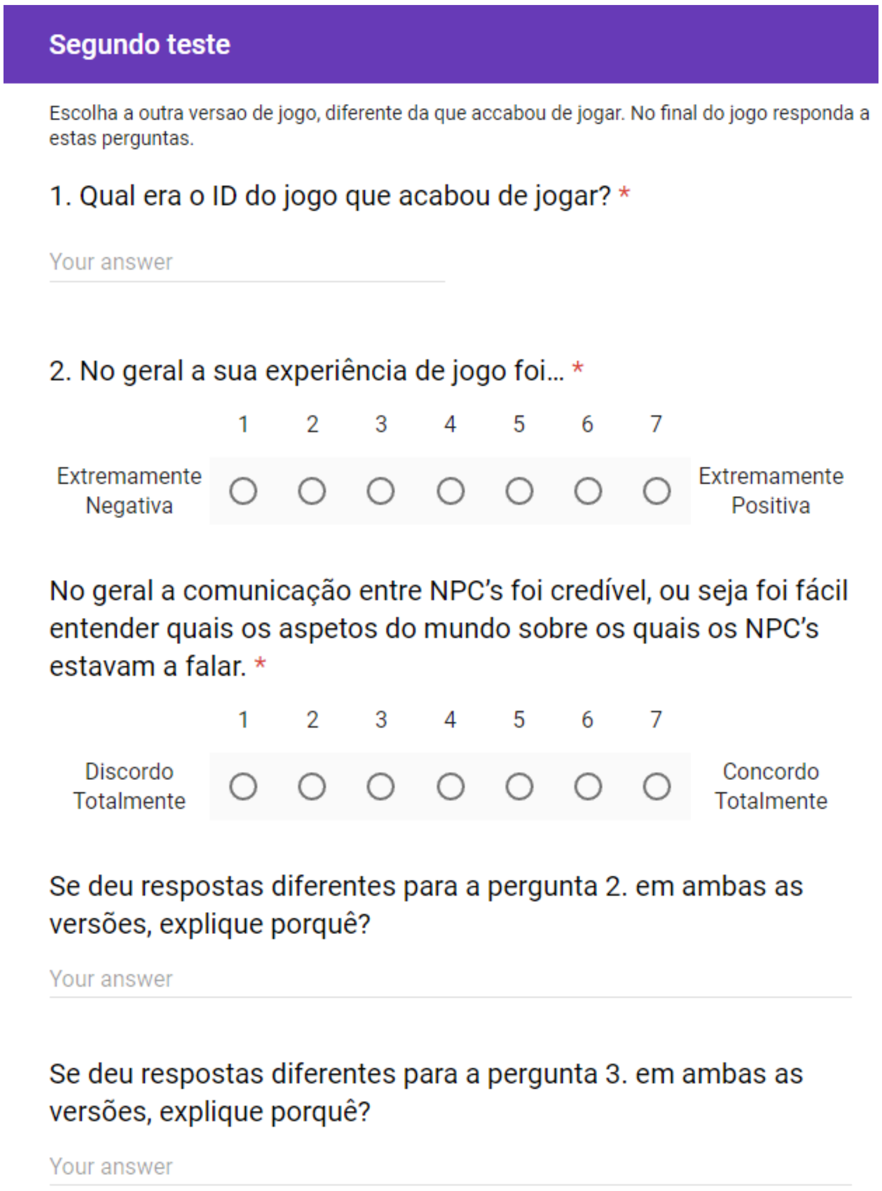
2. No geral a sua experiência de jogo foi... *

	1	2	3	4	5	6	7	
Extremamente Negativa	○	○	○	○	○	○	○	Extremamente Positiva

No geral a comunicação entre NPC's foi credível, ou seja foi fácil entender quais os aspetos do mundo sobre os quais os NPC's estavam a falar *

	1	2	3	4	5	6	7	
Discordo Totalmente	○	○	○	○	○	○	○	Concordo Totalmente

Figure A.2: Page 2 of the questionnaire. After the player tests the first version of the game he is asked the ID of the game he just played, his overall game experience (Likert scale of $1 = ExtremellyNegative \longrightarrow 7 = ExtremellyPositive$) and how credible was the communication between NPCs and if it was easy for the player to understand what the NPCs were talking about (Likert scale of $1 = TotallyDisagree \longrightarrow 7 = TotallyAgree$).



Segundo teste

Escolha a outra versao de jogo, diferente da que accabou de jogar. No final do jogo responda a estas perguntas.

1. Qual era o ID do jogo que acabou de jogar? *

Your answer

2. No geral a sua experiência de jogo foi... *

	1	2	3	4	5	6	7	
Extremamente Negativa	○	○	○	○	○	○	○	Extremamente Positiva

No geral a comunicação entre NPC's foi credível, ou seja foi fácil entender quais os aspetos do mundo sobre os quais os NPC's estavam a falar. *

	1	2	3	4	5	6	7	
Discordo Totalmente	○	○	○	○	○	○	○	Concordo Totalmente

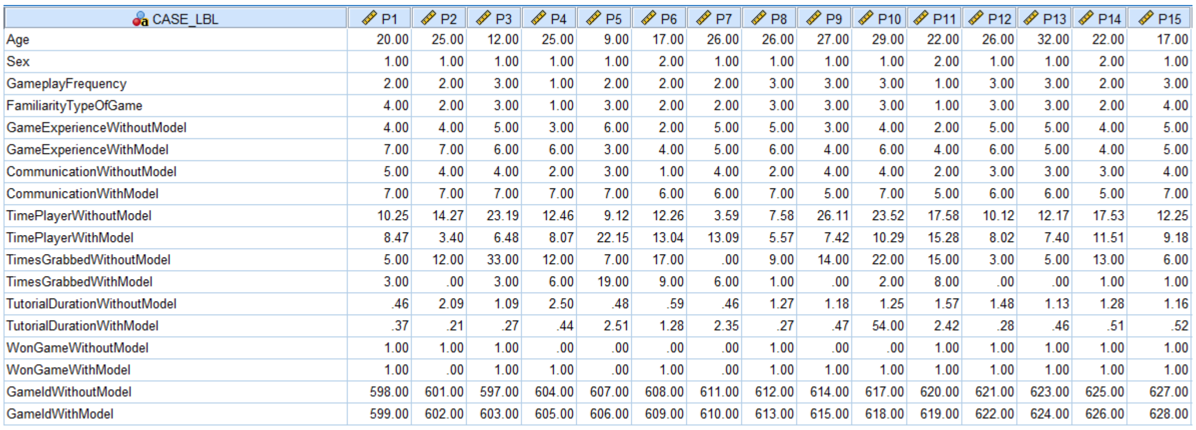
Se deu respostas diferentes para a pergunta 2. em ambas as versões, explique porquê?

Your answer

Se deu respostas diferentes para a pergunta 3. em ambas as versões, explique porquê?

Your answer

Figure A.3: Page 3 of the questionnaire. After the player plays the second version he responds to the same questions as described on page 2. If he gave different ratings to the same question on both versions we ask why did he gave different answers in order further understand the impact of our model on his gameplay.



CASE_LBL	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Age	20.00	25.00	12.00	25.00	9.00	17.00	26.00	26.00	27.00	29.00	22.00	26.00	32.00	22.00	17.00
Sex	1.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	2.00	1.00
GameplayFrequency	2.00	2.00	3.00	1.00	2.00	2.00	2.00	3.00	3.00	3.00	1.00	3.00	3.00	2.00	3.00
FamiliarityTypeOfGame	4.00	2.00	3.00	1.00	3.00	2.00	2.00	3.00	3.00	3.00	1.00	3.00	3.00	2.00	4.00
GameExperienceWithoutModel	4.00	4.00	5.00	3.00	6.00	2.00	5.00	5.00	3.00	4.00	2.00	5.00	5.00	4.00	5.00
GameExperienceWithModel	7.00	7.00	6.00	6.00	3.00	4.00	5.00	6.00	4.00	6.00	4.00	6.00	5.00	4.00	5.00
CommunicationWithoutModel	5.00	4.00	4.00	2.00	3.00	1.00	4.00	2.00	4.00	4.00	2.00	3.00	3.00	3.00	4.00
CommunicationWithModel	7.00	7.00	7.00	7.00	7.00	6.00	6.00	7.00	5.00	7.00	5.00	6.00	6.00	5.00	7.00
TimePlayerWithoutModel	10.25	14.27	23.19	12.46	9.12	12.26	3.59	7.58	26.11	23.52	17.58	10.12	12.17	17.53	12.25
TimePlayerWithModel	8.47	3.40	6.48	8.07	22.15	13.04	13.09	5.57	7.42	10.29	15.28	8.02	7.40	11.51	9.18
TimesGrabbedWithoutModel	5.00	12.00	33.00	12.00	7.00	17.00	.00	9.00	14.00	22.00	15.00	3.00	5.00	13.00	6.00
TimesGrabbedWithModel	3.00	.00	3.00	6.00	19.00	9.00	6.00	1.00	.00	2.00	8.00	.00	.00	1.00	1.00
TutorialDurationWithoutModel	.46	2.09	1.09	2.50	.48	.59	.46	1.27	1.18	1.25	1.57	1.48	1.13	1.28	1.16
TutorialDurationWithModel	.37	.21	.27	.44	2.51	1.28	2.35	.27	.47	54.00	2.42	.28	.46	.51	.52
WonGameWithoutModel	1.00	1.00	1.00	.00	.00	.00	.00	1.00	.00	.00	1.00	1.00	1.00	1.00	1.00
WonGameWithModel	1.00	.00	1.00	1.00	.00	1.00	.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
GameIdWithoutModel	598.00	601.00	597.00	604.00	607.00	608.00	611.00	612.00	614.00	617.00	620.00	621.00	623.00	625.00	627.00
GameIdWithModel	599.00	602.00	603.00	605.00	606.00	609.00	610.00	613.00	615.00	618.00	619.00	622.00	624.00	626.00	628.00

Figure A.4: Data obtained from the testing done on participants