Project planning and current technical structure

Cees-Jan Nolen, Robert Kraaijeveld, Steven Schenk

28-9-16

Contents

1 Introduction 2

2 Our frameworks and languages 3

3 Our personality model 4

4 General System Structure 5

5 NPC behavior 6
5.1 Actions and graph creation algorithm 6
5.2 Events 8

6 Emotions 9

1 Introduction

This document will explain the current technical configuration and structure for our internship project concerning procedural avatar generation.

A quick recap of what our project is aiming to achieve: We want to create a game for the Experience Room which contains procedurally generated avatars, with each avatar possessing a personality. The avatars should then perform certain actions or respond to situations according to its' personality. The player will be able to walk amongst the NPC population and will be presented various tasks, most of them revolving around recognizing which avatars exhibit which personality traits (For example: Select the avatar who you think is the most egoistic.).

The data resulting from these experiments could then be used for further study.

2 Our frameworks and languages

For this project, we are using a combination of Unity and the included C Sharp programming language, combined with the Casanova Programming Language. We use XML files to store settings such the possible actions that an NPC can take. Casanova is a very high level language, capable of easily manipulating game states. It uses a proxy system, consisting of C Sharp classes, to import Unity GameObjects and the like without having to deal with the relatively low level Unity code. The below image gives a visual example of this system.

As mentioned earlier, we use various XML files to store information or settings relative to our game. In order to make editing these files (specifically, the XML file which actions an NPC can take) easier, especially for non-programmers, we have created a simple GUI for editing these files, which runs locally in the users browser. This enables the user to easily edit the game settings without even having to pause the game. We implemented this editor in python, using the lightweight 'bottle' framework.

3 Our personality model

Currently, we are using primary factors 1-4 of the big 5 personality model to provide our NPC's with personality values. To recap, the primary factors of the big 5 personality model are as follows:

1. Extraversion vs. Introversion
2. Agreeableness (Which we use under the name 'Egoism vs. Altruism')
3. Thoroughness vs. Conscientiousness
4. Emotional Stability vs. Neuroticism

The 5th factor is 'Openness to experience' which we found nearly impossible to implement as concrete game behavior: Therefore we have forfeited its use for now.

Note that when we model these values to an NPC, they receive 8 values instead of 4: One value for each extreme of the factors, with the values ranging from 0 up to and including 100. For instance:

- Extraversion: 60
- Introversion: 40
- Etc.

In the future, we will likely add more complex personality models: More on the extendability of our current setup later.

4 General System Structure

The below ER Diagram documents the high level structure of our system:

As you can see, the C Sharp code is mainly responsible for setting up the appropriate datastructures and algorithms enabling an NPC to make decisions based on its personality, whilst Casanova is only responsible for translating the resulting list of actions into actual in-game behavior. We made this decision because Casanova is very good at directly manipulating the game state, and because it produces source code which is very readable to outsiders. Also, because Casanova is platform-independent, our game logic can be directly imported into other game frameworks. Contrary to C Sharp however, Casanova is not very usefull for creating large algorithms and datastructures: It has no mechanic for declaring functions or methods outside of game rules, and it only supports a very small number of datastructures natively.

To enable Casanova to access the properties of an NPC within Unity without having to deal with low-level Unity code, the results of the algorithms and datastructures defined in the C Sharp code are sent to a class which acts as a proxy from the C Sharp / Unity code to Casanova. Casanova can then access attributes of an individual NPC through instances of this proxy class.

5 NPC behavior

In our project, each NPC gets his own collection of personality factors and assorted values ranging from 0 to 100, like we talked about earlier. They also possess an 'emotionality threshold', which will be used to determine whether the NPC will display his behavior in a very emotional way, or not. Finally, they have a collection of accumulated personality values, which use the same 8 factors as their earlier mentioned personality values. These accumulated values will come into play in the decision making algorithm. Initially, these accumulated values are all zero, but they are summed with the personality modifiers of each actions that the NPC chooses to take: This process will be explained in more detail later.

NPC's can display their personalities to the player in one of three ways:

1. Through actions that they decide to execute because these actions' personality values are proportionate to their own.
2. If their emotionality threshold is exceeded, they will display very emotional behavior during the execution of actions. How exactly they display this emotion will depend on their personality values.
3. Through events: occurences which will trigger an immediate reaction from the NPC.

Up next, we will discuss each of these kinds of behavior in detail.

5.1 Actions and graph creation algorithm

We define an XML file containing a list of actions. Each action has the following properties:

1. Action name
2. Animation name
3. Position of action
4. A list of modifiers

The first three properties speak for themselves, but the fourth property deserves some more detailed explanation. The list of modifiers is a list consisting of the same 8 personality-extremes as an NPC possesses; These values are used later in the graph creation algorithm.

Now that we have defined a series of actions, we create an initial weighted, undirected graph as a datastructure for these actions. This graph does however not permit visiting the same node twice, since this could potentially result in an NPC constantly repeating the same action. Also, note that since every NPC has different personality values, they each get their own weighted graph of actions. This is done by calculating the dot product of an action's modifier value and the appropriate personality value for the NPC. This process is repeated for each personality value of the NPC, producing a total weight for the given action. Done for each action, this algorithm produces a weighted graph of all the actions for a specific NPC.

Very important to note is that this first weighted graph, and only this first graph, uses the fixed (!) personality values of the NPC: Further weighted graphs use the accumulated values which are modified by the actions that an NPC takes.

After the initial graph has been created and the NPC is spawned in the gameworld, a random action node is selected as a starting node. The starting node is added to the list of actions which will be passed to Casanova, and the NPC's accumulated values are summed with the action's modifier values. After that, the decision making algorithm marks the current node as 'visited', chooses the neighbour of the current action with the highest weight, sets it to be the new current action and adds it to the list of actions. The process then starts again with the new current node as starting node.

Note that given enough time, this algorithm will add each action in the graph to the list: This problem is solved in the following paragraph.

Every 10 seconds of game-time Casanova requests a fresh list of actions from this algorithm. The Unity proxy responds to this request by re-creating the weighted graph, using the accumulated values from the previous actions instead of the fixed personality values of a NPC, like we mentioned earlier. This constant refreshing ensures two things:

Firstly, that Casanova only executes the first few actions in the resulting actions list, so that the NPC does not execute each piece of behavior in the graph (Which is not desirable, since then it would be impossible to see what kind of personality a given NPC possesses.). Secondly, that an NPC's experiences changes his later behavior. The 10 second time-mark can of course be altered, and we are still experimenting with different time thresholds in order to create the most realistic behavior.

5.2 Events

In order to make our game environment more lifelike, we want to use events: Occurences that prompt an instance reaction from NPC's near them. Examples of this could be alarms sounding, NPC's fainting or more happy events like confetti suddenly appearing. This will provide some diversion to the game, as well as provide the player with a much more clear way to spot NPC personalities.

Events are, in their most basic form, radiuses. When an NPC enters an event radius or happens to be there when an event is spawned, the NPC will instantly be prompted to react to the event.

We decided to create a separate collection of event-actions in order to allow an NPC to react in a different way than usual to an event. This makes sense considering an NPC's reaction to an alarm sounding would not be to grab him/herself a cup of coffee. Since the algorithms discussed above are relatively generic, we can easily re-use these algorithms to create and weigh a graph of event-actions.

Each event has the following properties

1. A name and an Id.
2. Multiple variables concering the 'look and feel' of the event, including sounds and animations that should be played.
3. A duration.
4. An origin position.
5. A radius around this origin position.

When an NPC comes close enough to an event, he/she will react to it in a way that matches the personality values of said NPC. This is accomplished in the following way:

In the Casanova code, we spawn a random event. This is done after a random time interval, making predicting when events will happen impossible for the player. Each NPC has a reference to the current active event (or, list, if there are multiple active events). When an NPC enters the radius of an event, the algorithm that we discussed in the last chapter will change its' input nodes from normal action nodes, like getting coffee, to ' event action nodes'; actions that can only be performed when an NPC is near an active event.

This transition is done instantly, so that the NPC does not casually finish drinking his coffee before reacting to the fainted individual next to him. Also, since event action

nodes use exactly the same kind of XML variables as normal action nodes, the same algorithm can be used for both categories of actions.

The duration property of the event determines, naturally, how long the event will be active.

6 Emotions

7 Player interaction

8 Other improvements

The system as it currently stands has a lot of area for improvement. We still do not have any real art assets or character animations, making the NPC's look more like robotos than persons. These cosmetic Personality and emotions are in our opinion of less importance than the bigger issue of NPC design. In a real person personality and emotions are, of course, not the only factors that influence that persons' behavior in a given situation. Upbringing, inherited traits, culture, social standing, group dynamic etc are all very important factors when trying to decipher why someone exhibits a certain kind of behavior. Next up we will discuss the feasibility of including these factors within our simulation of human behavior.

Including upbringing and inherited traits as factors within our virtual decision-making process would require two NPC's to combine their 'behavioral factors' in order to create a new NPC; while we consider this a very interesting subject, we believe it to be out of scope for this project.

Introducing an NPC's culture into the mix could be a very valuable asset, if we represent cultural influence accurately enough. Having an accurate representation of the impact of culture on behavior could make this project into a training tool for international businesses, since unknowingly disrespecting a clients culture could have disastrous consequences. Important cultural differences lie mostly in areas such as personal space, appropriate ways of approaching a stranger etc. These kinds of interactions could be easily implemented by having NPC's react to the players' immediate presence in a way appropriate to their culture.

Last but certainly not least, we will discuss the potential implementation of social standing and group dynamics within our project. Within psychology, groups are classified to be either a formal or informal group. Each of these categories can be further defined as either a command group, an interest group or a friendship group. Broadly speaking, formal command and interest groups are usually found within the work environment, while informal friend groups are more commonly found within clubs and friend groups.

Within the context of our project, we think informal friend groups are a good starting point.The other types of groups, especially the more formal types, always exist because of a higher, shared purpose which the NPC's in our system do not possess. Informal friend groups however (We will refer to them as IFG's for short) only exist because their members have shared interests or share certain personality traits. Especially the last kind of reason for an IFG to exist would be easy to implement within our system, since we already model our NPC's personalities. We could easily let NPC's with relatively proportionate personality values flock together one by one into a loose group. If we decide to dig deeper into the concept of group formation, we might also want to include the different stages of group formation that are known in psychology literature:

A big challenge when implementing groups of NPC's is that of each individual NPC's autonomy level: How much does an individual decide for him/herself whilst being a member of a group, and how much does the NPC follow the leading group behavior? We will likely implement this with a chance function, in which the level of autonomy being given to each group member determines their chance of acting according to their own personality values, rather than follow majority group decisions. This problem might be made easier if we limit NPC groups to NPC's with nearly identical personality values only: In this case, the homogenous personality values will ensure that most group-action decisions are made unanimously most of the time.