Project planning and current technical structure

Cees-Jan Nolen, Robert Kraaijeveld, Steven Schenk

28-9-16

Contents

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

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.