ExtremeAI 2 User Manual

Unity-specific version
rev 2.0, March 2017

Table of Contents

Introduction ii

1.0 Installation and Overview 1
- 1.1 System requirements 1
- 1.2 Initial setup for Unity 1

2.0 Using the Engine 2
- 2.1 Setting up and editing characters 2
- 2.2 Using the character personalities in a game 11

3.0 Examples 18
- 3.1 Remember to set the NPC and player names! 18
- 3.2 Getting it all started: using Init 18
- 3.3 Changing speed of change 19
- 3.4 Using AIReturnResult and AINoResult 21

4.0 Support and Credits 24

5.0 Citations 25

Appendix 26

Introduction

Welcome to ExtremeAI 2, Unity edition! ExtremeAI provides a simple but complete add-on to create and change non-player personalities both before and during a game. With such power, your characters will react more realistically and will change their opinions of players depending on how they are treated. You can create different personalities for as many NPCs as you wish, and each will develop independently of all the others, with potentially different reactions depending on interactions with a player, with other players and NPCs, and even with world events.

Through a set of menus in the Unity editor, you can easily create and edit character personalities and assign them to NPCs. And through scripting, you can access these personalities in-game and have your NPCs react, grow, and change throughout the game's run, giving every play-through a different feel, giving your players that much more incentive to return to your game and play it again, wondering what would happen if they had treated those soldiers, or shopkeepers, or henchmen a little differently …

New to Version 2
ExAI v2 is different from v1 in the following ways:

• It is much faster in-game, able to deal with large numbers of personality checks much more efficiently than in v1
• New character personalities can be created on the fly, in-game
• Character personalities can also be deleted in-game
• You can now check and change more personality values directly in code, such as changing facet values directly
• A new drop-down list in the character editor lets you see all the character names you've created at once, rather than offering suggestions
• You can see all the registered players at once, too, through the Register Player screen
• You can also retrieve lists of all character personalities and players through code
• Two new response types have been added having to do with voting, as in the TheyVote! demo
• You can now choose the save location for character personalities, rather than being forced to use the default

All character personalities created in version 1 will work in version 2. Some of the method calls are slightly different, however, so you'll need to adjust v1 code to v2. See the method descriptions in this manual.

Please feel free to contact us at Quantum Tiger Games with questions, or even just to share your experiences using ExtremeAI, at *support@quantumtigergames.com*.

Thank you!
Jeff Georgeson
Quantum Tiger Games, LLC

1.0 Installation and Overview

1.1 System requirements

The Unity-specific version of ExtremeAI 2 works in Unity 5.x. The disk space, memory, etc. requirements are minimal.

1.2 Initial setup for Unity

It’s easy to set up the ExtremeAI engine! There are two ways, depending on how you downloaded the engine—through the asset store or as a zip file.

If you bought ExtremeAI in Unity's asset store, simply follow the import instructions there. The dlls and resources will be imported into their appropriate folders automatically.

Make sure to read the notes after the "That's it!" paragraph below.

If you have a zip file, first, unzip the ExtremeAI file (if you haven’t already; if you’re reading this manual, you probably have, come to think of it ...).

Next, copy the ExtremeAI folder into your Assets folder. The ExtremeAI folder contains three subfolders: Editor, Resources, and Scripts.

Finally, while some older versions of Unity required that you copy the contents of ExtremeAI’s Scripts subfolder (**TaiPE_Lib_v2.dll** or **TaiPE_Lib_Light_v2.dll**) into whichever folder you’re using for your project’s script files, newer versions allow you to leave these dlls where they are. Just FYI.

That’s it!

Actually, that’s almost it ... you may need to click on the menu bar in Unity to get the Tools/ExtremeAI menu option to show up for the first time.

Also, to successfully build a project using ExtremeAI, you’ll need to change the PlayerSettings in Unity (Edit->Project Settings->Player, or click Player Settings from the Build Settings dialogue box) setting for API Compatibility Level (at the bottom, under Optimization) to .NET 2.0 (NOT the default “.NET 2.0 Subset”). The subset lacks the functionality necessary to read some of the dlls used after compiling.

And that’s really it! You’re ready to build personalities into your NPCs!

2.0 Using the Engine

2.1 Setting up and editing characters

The various submenus of the ExtremeAI menu (found in Tools->ExtremeAI) allow you to create and edit the base personalities for your NPCs. This is all pregame stuff; the characters will begin your game in whatever configuration you choose here.

2.1.2 Background

You can create personalities for as few or as many NPCs as you wish; there is no requirement that every NPC in your game use the ExtremeAI engine. We have endeavored to make the creation process as simple as possible, however, giving you the option of using preset base personalities or tweaking as many as 39 different aspects of your characters' personalities. (Note that in ExtremeAI Light, you can tweak only 11 aspects.)

These aspects are the same stimuli/responses you will be able to choose from your game code. For instance, you can adjust the initial intensity of an NPC's "kind"ness—that is, his likelihood of responding to kindness with kindness, or to perform a kind act—using the character editor. Note that you are not directly changing the underlying facets of the NPC's personality (there is no "kindness" facet, for instance), but are affecting the underlying facets that go into making us more prone to kindness. Because of this, when you change kind, you affect the other stimuli/response types that have similar facets as components.

Also note that the numerical values you see are for your guidance only; the underlying personality facets are the core of the engine, and are ultimately used (along with our personality algorithms) to choose/change levels of response in the actual game situation.

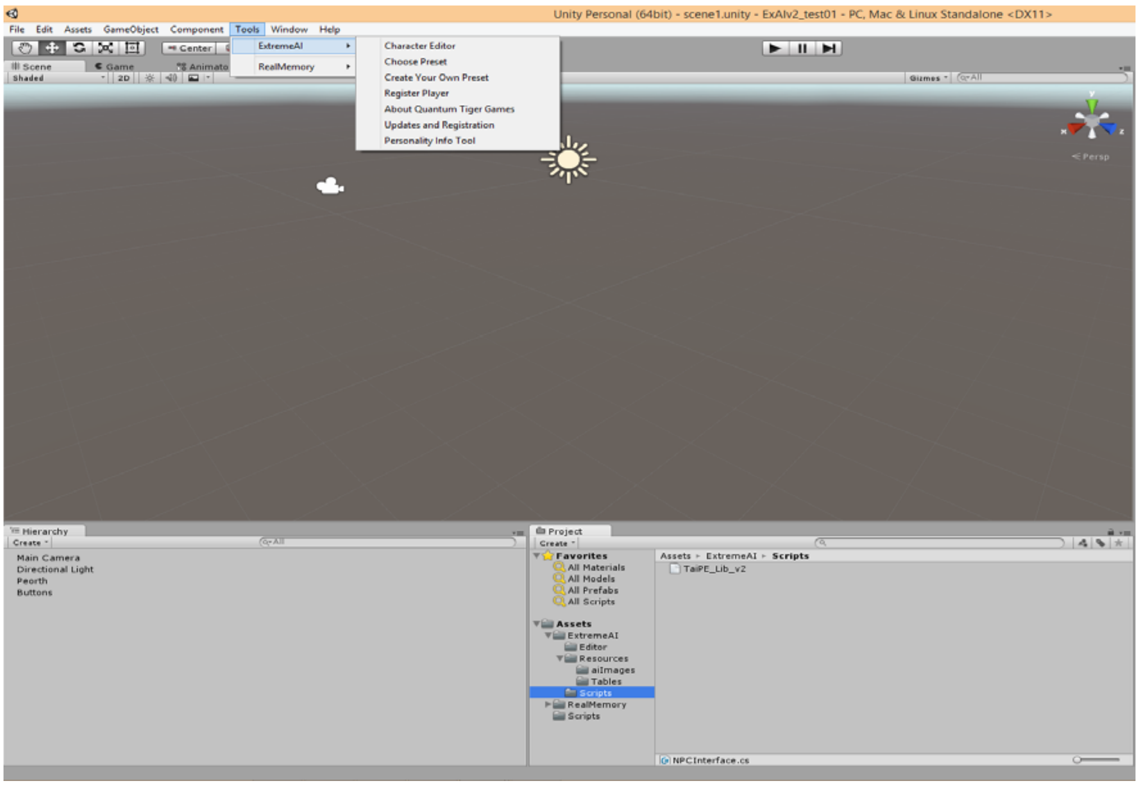


Figure 1 Unity editor screen, including ExtremeAI menu.

2.1.3 What are these “stimulus/response types”?

Personality theory in general posits that there are certain base elements that go into creating a person’s personality, and that these factor in various ways into the sorts of decisions and actions we make in our lives. In the Five Factor Model, a well-respected theory (see, for instance, Goldberg, 1993; Costa & McCrae, 1995; John et al, 2010; McCrae & Costa, 2010; DeYoung, 2010*), these elements are the 30 personality facets listed in Table 1, which are grouped into five overall categories (hence the ‘Five’ in the name of the model).

Rather than having the developer try to check the individual personality facets and figure out how these might apply to various everyday stimuli (such as another person being nice to you) or response types (like whether an NPC will go up to a stranger and be nice without provocation), the engine provides 39 possible stimuli/responses (see Table 2) that have already been keyed into the various facets (using our own extensive research and applying this to game-type situations). Thus the developer merely has to query whether a stimulus will elicit a strong reaction, and the engine will take care of not only finding that answer but figuring out how much (if at all) having that interaction affects the personality of the NPC in future interactions, either with the same player or with other players (or NPCs or, even, world events). Thus, just as in real life, the very act of interacting with others can have an effect on how a character deals with the future, and on how similar stimuli are dealt with.

New to ExAI2: Two new response types, **voteType** and **willVote**, have been added. These allow the developer to determine a voter’s conservative-liberal leanings and whether the voter is likely to actually vote. A lower score in voteType indicates general conservatism; a high score indicates a more liberal character. With willVote, the higher the score, the more likely the character is to vote.

Also as in real life, having high tendencies toward a certain reaction does not mean the character will react in the same exact way each and every time; our algorithms take this into account and return a reaction intensity from which you can make your own judgment as to what the NPC says or does.

Openness	**Extraversion**	**Neuroticism**
1. Fantasy	13. Warmth	25. Anxiety
2. Aesthetics	14. Gregariousness	26. Angry Hostility
3. Feelings	15. Assertiveness	27. Depression
4. Actions	16. Activity	28. Self-Consciousness
5. Ideas	17. Excitement Seeking	29. Impulsiveness
6. Values	18. Positive Emotions	30. Vulnerability
Conscientiousness	**Agreeableness**	
7. Competence	19. Trust	
8. Order	20. Straightforwardness	
9. Dutifulness	21. Altruism	
10. Achievement Striving	22. Compliance	
11. Self-Discipline	23. Modesty	
12. Deliberation	24. Tender-Mindedness	

Table 1 The Five Categories and Their Facets (categories in bold)

affectionate	excitability	orderly
ambitious	gregarious	productive
anger	guilt	quarrelsome
annoying	happiness	reluctance
anxiety	helpfulness	sadness
assertiveness	humour	selfishness
condescension	imaginative	standoffish
conformity	impulsive	sternness
decitfulness	intellectual	stubborn
demanding	intimidating	talkative
dependability	jealousy	voteType
distrustful	kind	willVote
efficiency	moodiness	wittiness

Table 2 Possible Stimuli/Response Types (to be queried by the developer)

2.1.4 ExtremeAI menu

See again Figure 1. Pretty self-explanatory. From here you choose:

Character Editor to create or edit an existing character using the full set of possible stimuli/response types—gives you access to the 39 different stimuli/response types (or 11 in the Light version). Use this to fine-tune a character. **New in version 2:** The drop-down list of characters shows you all the character personalities you've already set up.

Choose Preset shows you a list of preset personality types—allowing you to quickly make a "type" of character without having to adjust anything. You can, however, later edit these characters using the full set of stimuli/response types if you wish.

Create Your Own Preset—find yourself making a lot of similar characters, but none of our presets quite fits? This lets you create up to four of your own presets. (Not available in the Light version.)

Register Player—In order for your NPCs to keep track of their attitudes and feelings toward your player(s), they have to know how to reference him/her/them. You will need to use this menu to register the player(s). Note that any characters you create with the

Ultimately, of course, the extent to which you consult the personality engine is up to you; if you want a certain NPC to follow a very prescribed set of actions, that's fine; it does not interfere with the Engine's functioning to do this (although it won't be able to develop that NPC's personality, obviously). If an NPC has to give out a piece of important information, the Engine will not forbid it (how could it?); what you might do, however, is have the NPC's phrasing be different depending on how she feels about this player. Even subtle differences like this can make the gameplay even more interesting than it already is, can add a bit of mystery and intrigue and depth (especially depth) to the world and the characters and people in it, and ultimately lead to a more satisfying experience for the (presumably human) players playing your game.

* The full references can be found at the back of this manual, for those interested.

editor or presets will automatically be registered for you, so that they can interact with each other. **New in version 2:** You can see all registered players in the drop-down list labeled “Player List.”

About Quantum Tiger Games—gives us the opportunity to talk about ourselves.

Updates and Registration—gives you the opportunity to register Extreme AI, which gives you quick access to updates and other info.

Personality Info Tool—only available in the full version of Extreme AI. Allows you to see the underlying facet values for each character, rather than just the stimulus/response types.

2.1.5 Creating/editing characters

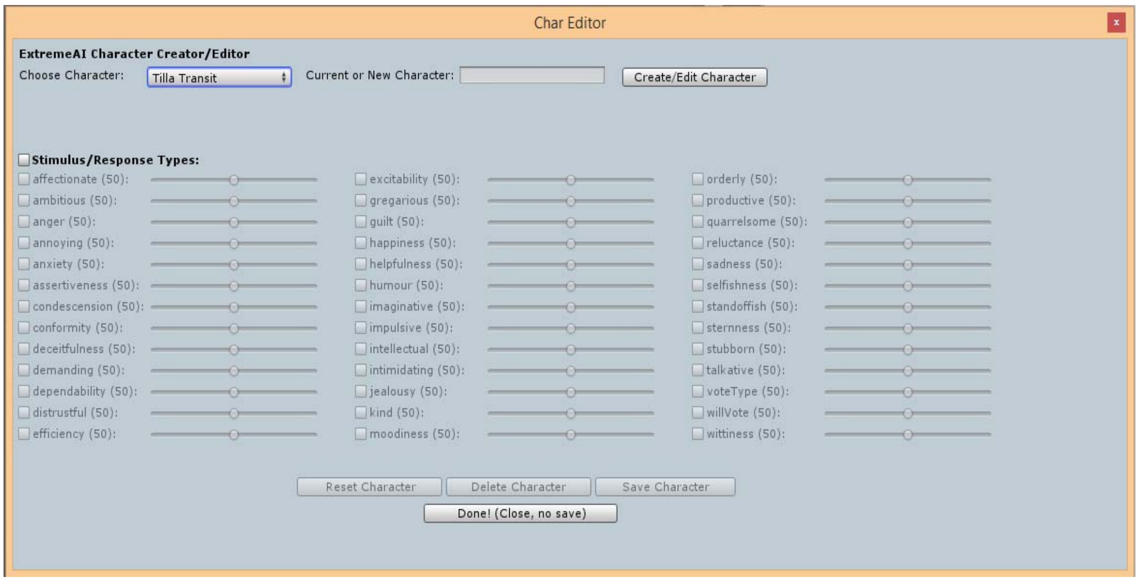
There are two ways to create a character: fine-tuning a character in detail, or using a preset to make a quick general type. For example, you could tailor a specific NPC to be more kind, intellectual, and imaginative (which will make that character more prone to wittiness and slightly less likely to anger as well, due to the underlying facets). You could also try the Professor preset, which will give you a basically intellectual character, but not one who is necessarily kind or imaginative. Characters created from a preset can be fine-tuned later using the character editor.

Following are detailed instructions for creating characters.

Character Editor: Create a character in detail
If you click the “Character Editor” menu, a window opens containing a drop-down list of existing characters, a text field for entering a new character name, and a set of sliders (see Figure 2, top).

Once you have selected/typed a name, press the Create/Edit Character button. This will activate the sliders, each set to a value between 0 (the character has an extremely low propensity for this) and 100 (the character reacts in this way very intensely) (see Figure 2, middle or bottom). Zero kindness would indicate a very unkind person; 100 would be someone who was almost saintly. If the character already exists, its current values from her/his facets will be calculated (Figure 2, middle); a new character will start out with all 50s (Figure 2, bottom). Note that the currently edited character name is in the text box.

To adjust the values, click the toggle box next to a response type, then move the slider up or down. When you release the slider, the character’s facets will be refigured to match the new response value, and then all his/her responses will be recalculated to match the new facets. For example, changing Newlin Newbie’s Affectionate score from 50 to 70 also increases his Kindness (to 57) but decreases his Anger (tendency to become angry) somewhat (to 46), along with changing several other values slightly. This is because the psychological factors underlying the Kindness response (the facets) were changed when you adjusted Kindness, and thus every other response based on any of the same factors changed as well (see Figure 3).


Char Editor

ExtremeAI Character Creator/Editor

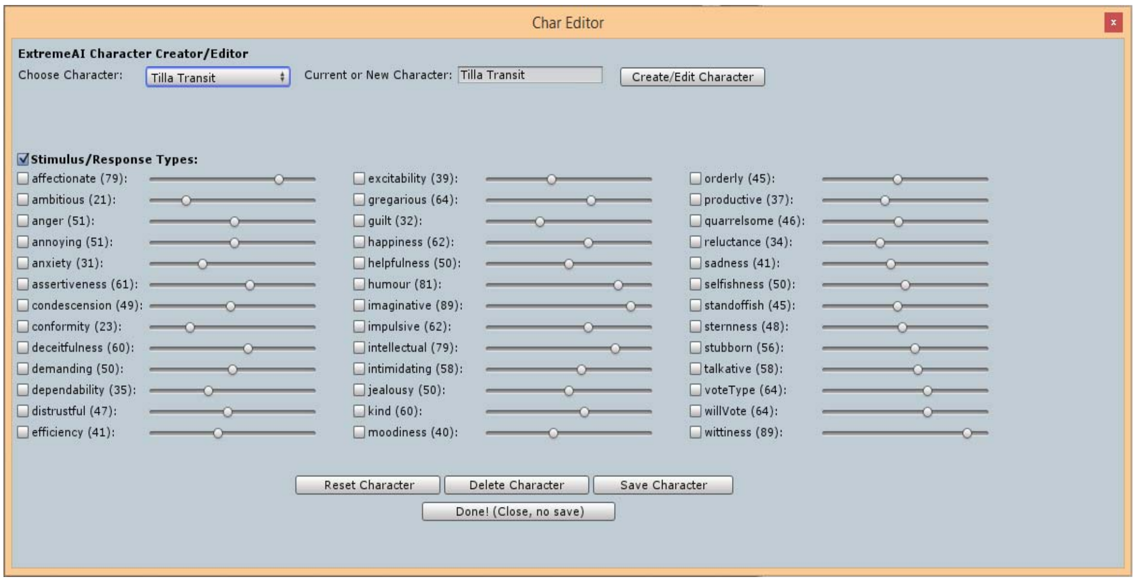
Choose Character: Tilla Transit Current or New Character: Create/Edit Character

☐ **Stimulus/Response Types:**

☐ affectionate (50):	☐ excitability (50):	☐ orderly (50):
☐ ambitious (50):	☐ gregarious (50):	☐ productive (50):
☐ anger (50):	☐ guilt (50):	☐ quarrelsome (50):
☐ annoying (50):	☐ happiness (50):	☐ reluctance (50):
☐ anxiety (50):	☐ helpfulness (50):	☐ sadness (50):
☐ assertiveness (50):	☐ humour (50):	☐ selfishness (50):
☐ condescension (50):	☐ imaginative (50):	☐ standoffish (50):
☐ conformity (50):	☐ impulsive (50):	☐ sternness (50):
☐ deceitfulness (50):	☐ intellectual (50):	☐ stubborn (50):
☐ demanding (50):	☐ intimidating (50):	☐ talkative (50):
☐ dependability (50):	☐ jealousy (50):	☐ voteType (50):
☐ distrustful (50):	☐ kind (50):	☐ willVote (50):
☐ efficiency (50):	☐ moodiness (50):	☐ wittiness (50):

Reset Character Delete Character Save Character

Done! (Close, no save)


Char Editor

ExtremeAI Character Creator/Editor

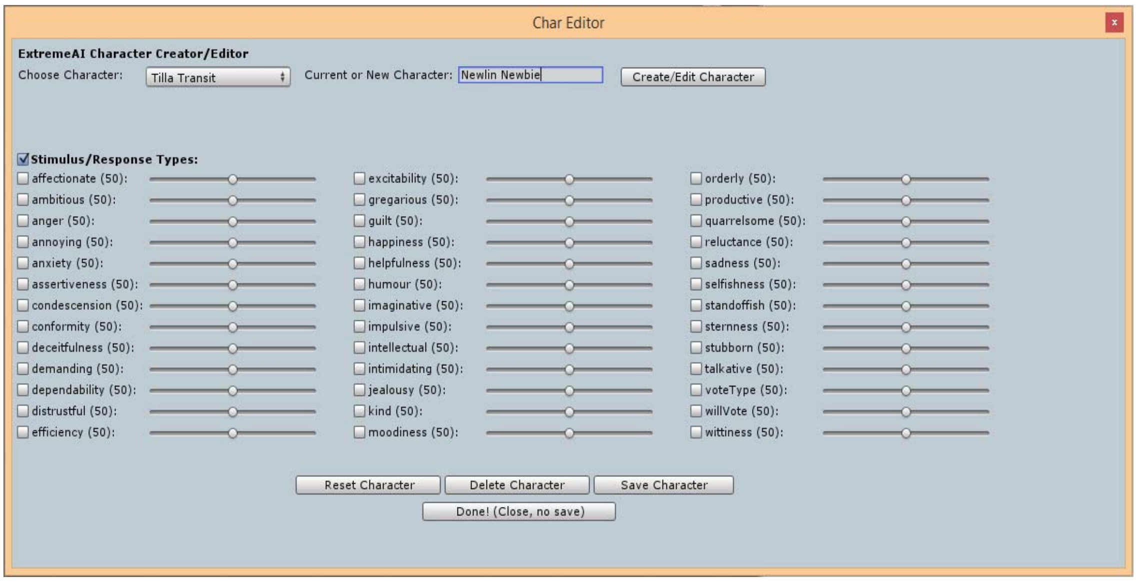
Choose Character: Tilla Transit Current or New Character: Tilla Transit Create/Edit Character

☑ **Stimulus/Response Types:**

☐ affectionate (79):	☐ excitability (39):	☐ orderly (45):
☐ ambitious (21):	☐ gregarious (64):	☐ productive (37):
☐ anger (51):	☐ guilt (32):	☐ quarrelsome (46):
☐ annoying (51):	☐ happiness (62):	☐ reluctance (34):
☐ anxiety (31):	☐ helpfulness (50):	☐ sadness (41):
☐ assertiveness (61):	☐ humour (81):	☐ selfishness (50):
☐ condescension (49):	☐ imaginative (89):	☐ standoffish (45):
☐ conformity (23):	☐ impulsive (62):	☐ sternness (48):
☐ deceitfulness (60):	☐ intellectual (79):	☐ stubborn (56):
☐ demanding (50):	☐ intimidating (58):	☐ talkative (58):
☐ dependability (35):	☐ jealousy (50):	☐ voteType (64):
☐ distrustful (47):	☐ kind (60):	☐ willVote (64):
☐ efficiency (41):	☐ moodiness (40):	☐ wittiness (89):

Reset Character Delete Character Save Character

Done! (Close, no save)


Char Editor

ExtremeAI Character Creator/Editor

Choose Character: Tilla Transit Current or New Character: Newlin Newbie Create/Edit Character

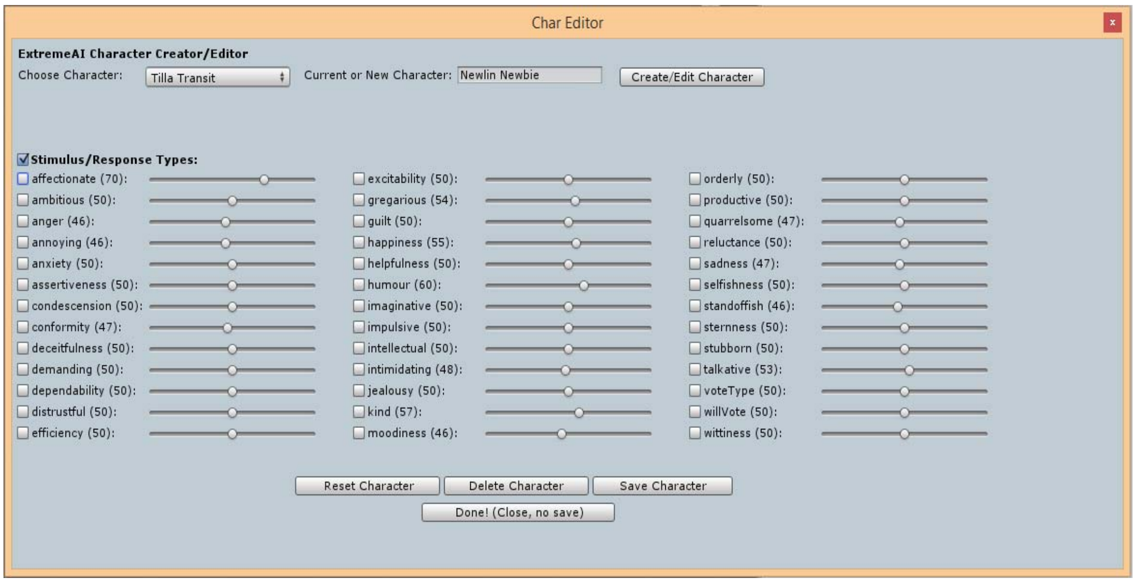
☑ **Stimulus/Response Types:**

☐ affectionate (50):	☐ excitability (50):	☐ orderly (50):
☐ ambitious (50):	☐ gregarious (50):	☐ productive (50):
☐ anger (50):	☐ guilt (50):	☐ quarrelsome (50):
☐ annoying (50):	☐ happiness (50):	☐ reluctance (50):
☐ anxiety (50):	☐ helpfulness (50):	☐ sadness (50):
☐ assertiveness (50):	☐ humour (50):	☐ selfishness (50):
☐ condescension (50):	☐ imaginative (50):	☐ standoffish (50):
☐ conformity (50):	☐ impulsive (50):	☐ sternness (50):
☐ deceitfulness (50):	☐ intellectual (50):	☐ stubborn (50):
☐ demanding (50):	☐ intimidating (50):	☐ talkative (50):
☐ dependability (50):	☐ jealousy (50):	☐ voteType (50):
☐ distrustful (50):	☐ kind (50):	☐ willVote (50):
☐ efficiency (50):	☐ moodiness (50):	☐ wittiness (50):

Reset Character Delete Character Save Character

Done! (Close, no save)

Figure 2 Create a detailed character. (top) Enter a character name, then (middle, character exists, or bottom, new character) edit her or his details.



ExtremeAI Character Creator/Editor

Choose Character: Tilla Transit Current or New Character: Newlin Newbie

☑ Stimulus/Response Types:

affectionate (70):	excitability (50):	orderly (50):
ambitious (50):	gregarious (54):	productive (50):
anger (46):	guilt (50):	quarrelsome (47):
annoying (46):	happiness (55):	reluctance (50):
anxiety (50):	helpfulness (50):	sadness (47):
assertiveness (50):	humour (60):	selfishness (50):
condescension (50):	imaginative (50):	standoffish (46):
conformity (47):	impulsive (50):	sternness (50):
deceitfulness (50):	intellectual (50):	stubborn (50):
demanding (50):	intimidating (48):	talkative (53):
dependability (50):	jealousy (50):	voteType (50):
distrustful (50):	kind (57):	willVote (50):
efficiency (50):	moodiness (46):	wittiness (50):

Figure 3 Newlin Newbie with adjusted stimuli/response values

When you are finished adjusting the values, click the Save Character button to save your edits. And if you're unhappy with your changes and would like to come back to this character later (or never), click Done. This closes the editor window without saving. You can also click Done without ever entering a character name; this closes the window without creating or editing anything.

NOTE: These settings create a base personality, applied initially to every interaction the character has with anyone and anything in the game world. As the character interacts with other individuals, her reactions toward those individuals will vary depending on what those interactions have been; in other words, she may grow to despise a player who has been continually nasty to her, yet love someone who has been nothing but kind and giving. Additionally, an NPC's base responses (to strangers, say) will be affected by all of her interactions with others and the world; an NPC who has been treated badly by almost everyone will grow to be cynical and distrustful, even if he started out a kind and genial soul.

And what do the other buttons on this screen do?

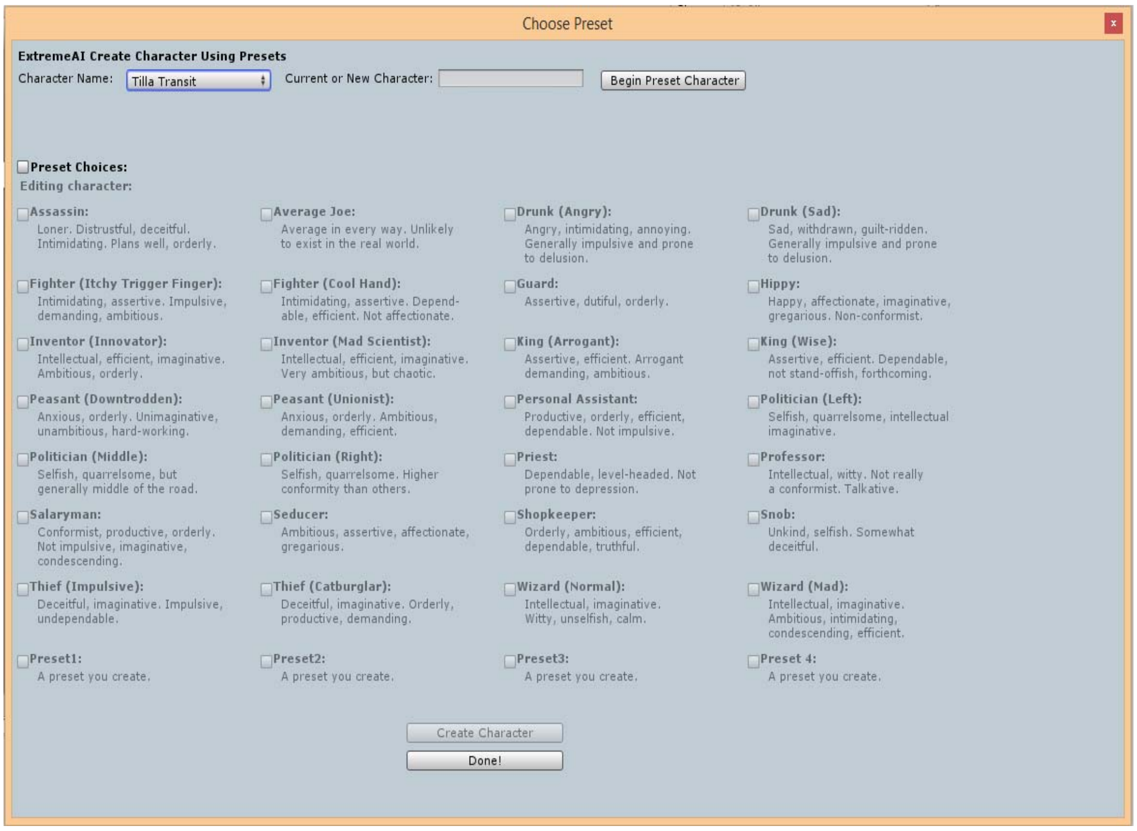
Reset Character—The Reset button resets a character's slider values to those he/she had when imported (or created, if new). New characters will revert to all average values.

Delete Character—Deletes the current character. Note that this is not reversible!

Choose Preset: Create a character using presets

Clicking ExtremeAI -> Choose Preset allows you to create a character quickly, without having to adjust slider values. There are 28 different presets (14 in the Light version), some of which are subtypes of the same general personality.

First, either select an existing character from the drop-down list or type a name for new


Choose Preset

ExtremeAI Create Character Using Presets

Character Name: Tilla Transit | Current or New Character: | Begin Preset Character

☐ **Preset Choices:**
Editing character:

☐ **Assassin:** Loner. Distrustful, deceitful. Intimidating. Plans well, orderly.	☐ **Average Joe:** Average in every way. Unlikely to exist in the real world.	☐ **Drunk (Angry):** Angry, intimidating, annoying. Generally impulsive and prone to delusion.	☐ **Drunk (Sad):** Sad, withdrawn, guilt-ridden. Generally impulsive and prone to delusion.
☐ **Fighter (Itchy Trigger Finger):** Intimidating, assertive. Impulsive, demanding, ambitious.	☐ **Fighter (Cool Hand):** Intimidating, assertive. Dependable, efficient. Not affectionate.	☐ **Guard:** Assertive, dutiful, orderly.	☐ **Hippy:** Happy, affectionate, imaginative, gregarious. Non-conformist.
☐ **Inventor (Innovator):** Intellectual, efficient, imaginative. Ambitious, orderly.	☐ **Inventor (Mad Scientist):** Intellectual, efficient, imaginative. Very ambitious, but chaotic.	☐ **King (Arrogant):** Assertive, efficient. Arrogant demanding, ambitious.	☐ **King (Wise):** Assertive, efficient. Dependable, not stand-offish, forthcoming.
☐ **Peasant (Downtrodden):** Anxious, orderly. Unimaginative, unambitious, hard-working.	☐ **Peasant (Unionist):** Anxious, orderly. Ambitious, demanding, efficient.	☐ **Personal Assistant:** Productive, orderly, efficient, dependable. Not impulsive.	☐ **Politician (Left):** Selfish, quarrelsome, intellectual imaginative.
☐ **Politician (Middle):** Selfish, quarrelsome, but generally middle of the road.	☐ **Politician (Right):** Selfish, quarrelsome. Higher conformity than others.	☐ **Priest:** Dependable, level-headed. Not prone to depression.	☐ **Professor:** Intellectual, witty. Not really a conformist. Talkative.
☐ **Salaryman:** Conformist, productive, orderly. Not impulsive, imaginative, condescending.	☐ **Seducer:** Ambitious, assertive, affectionate, gregarious.	☐ **Shopkeeper:** Orderly, ambitious, efficient, dependable, truthful.	☐ **Snob:** Unkind, selfish. Somewhat deceitful.
☐ **Thief (Impulsive):** Deceitful, imaginative. Impulsive, undependable.	☐ **Thief (Catburglar):** Deceitful, imaginative. Orderly, productive, demanding.	☐ **Wizard (Normal):** Intellectual, imaginative. Witty, unselfish, calm.	☐ **Wizard (Mad):** Intellectual, imaginative. Ambitious, intimidating, condescending, efficient.
☐ **Preset1:** A preset you create.	☐ **Preset2:** A preset you create.	☐ **Preset3:** A preset you create.	☐ **Preset 4:** A preset you create.

Figure 4 Create a character using presets.

character in the Character Name box (see Figure 4); this is the same as in the Create/Edit Character window. Click "Begin Preset Character" when ready.

After this the Preset Character options will become available (see Figure 5). Click the check box next to the desired preset, then click "Create Character". This creates the character.

At this point you can choose/type in another name to create another preset character, or you can press "Done" to close the window. Pressing "Done" before pressing "Create Character" closes the window and does NOT create the preset character.

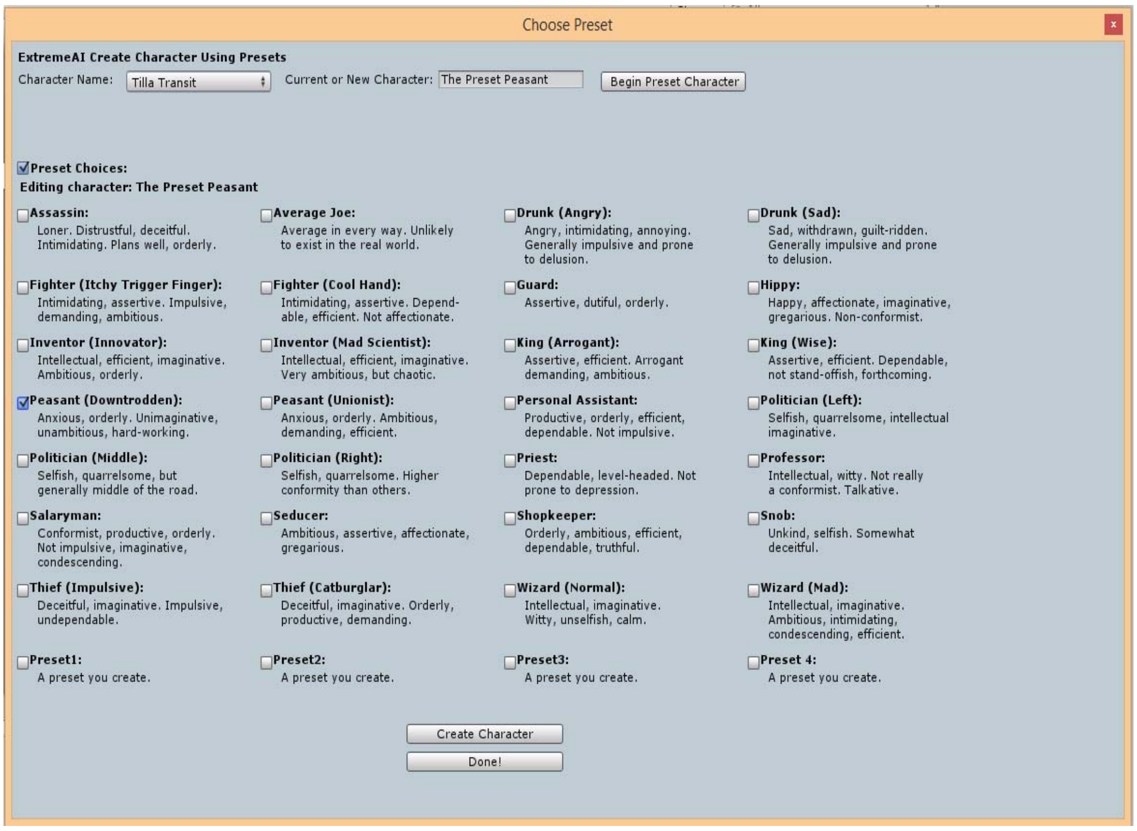
The last row of preset choices is all presets you can create and edit yourself, as covered next. (These are not available in the Light version.)

Create Your Own Preset: Your personal army of preset personalities

If none of the existing presets works for you, you can create up to four of your own (full version only). You do this by selecting Create Your Own Preset from the ExtremeAI menu.

In the window that comes up, you will be given four options from which to choose (see Figure 6); these are the current user-created presets (or the defaults if you haven't created any). Double-click on one of these to select it for editing.

Once selected, the name of the preset will appear in the "New Name (if desired)" box


Choose Preset

ExtremeAI Create Character Using Presets

Character Name: Tilla Transit   Current or New Character: The Preset Peasant   Begin Preset Character

☑ Preset Choices:

Editing character: The Preset Peasant

☐ **Assassin:** Loner. Distrustful, deceitful. Intimidating. Plans well, orderly.	☐ **Average Joe:** Average in every way. Unlikely to exist in the real world.	☐ **Drunk (Angry):** Angry, intimidating, annoying. Generally impulsive and prone to delusion.	☐ **Drunk (Sad):** Sad, withdrawn, guilt-ridden. Generally impulsive and prone to delusion.
☐ **Fighter (Itchy Trigger Finger):** Intimidating, assertive. Impulsive, demanding, ambitious.	☐ **Fighter (Cool Hand):** Intimidating, assertive. Dependable, efficient. Not affectionate.	☐ **Guard:** Assertive, dutiful, orderly.	☐ **Hippy:** Happy, affectionate, imaginative, gregarious. Non-conformist.
☐ **Inventor (Innovator):** Intellectual, efficient, imaginative. Ambitious, orderly.	☐ **Inventor (Mad Scientist):** Intellectual, efficient, imaginative. Very ambitious, but chaotic.	☐ **King (Arrogant):** Assertive, efficient. Arrogant demanding, ambitious.	☐ **King (Wise):** Assertive, efficient. Dependable, not stand-offish, forthcoming.
☑ **Peasant (Downtrodden):** Anxious, orderly. Unimaginative, unambitious, hard-working.	☐ **Peasant (Unionist):** Anxious, orderly. Ambitious, demanding, efficient.	☐ **Personal Assistant:** Productive, orderly, efficient, dependable. Not impulsive.	☐ **Politician (Left):** Selfish, quarrelsome, intellectual imaginative.
☐ **Politician (Middle):** Selfish, quarrelsome, but generally middle of the road.	☐ **Politician (Right):** Selfish, quarrelsome. Higher conformity than others.	☐ **Priest:** Dependable, level-headed. Not prone to depression.	☐ **Professor:** Intellectual, witty. Not really a conformist. Talkative.
☐ **Salaryman:** Conformist, productive, orderly. Not impulsive, imaginative, condescending.	☐ **Seducer:** Ambitious, assertive, affectionate, gregarious.	☐ **Shopkeeper:** Orderly, ambitious, efficient, dependable, truthful.	☐ **Snob:** Unkind, selfish. Somewhat deceitful.
☐ **Thief (Impulsive):** Deceitful, imaginative. Impulsive, undependable.	☐ **Thief (Catburglar):** Deceitful, imaginative. Orderly, productive, demanding.	☐ **Wizard (Normal):** Intellectual, imaginative. Witty, unselfish, calm.	☐ **Wizard (Mad):** Intellectual, imaginative. Ambitious, intimidating, condescending, efficient.
☐ **Preset1:** A preset you create.	☐ **Preset2:** A preset you create.	☐ **Preset3:** A preset you create.	☐ **Preset 4:** A preset you create.

Create Character

Done!

Figure 5 The Preset Choices

and the sliders will become available. To rename the preset, type a new name in the box and click “Change Preset Name”. To change the preset’s stimulus/response values, click the toggle box next to a response type, then move the slider left or right. When you release the slider, the preset’s facets will be refigured to match the new response value, and then all its responses will be recalculated to match the new facets (just as when using the Character Create/Edit screen). The preset’s values are changed in the database live as you change them. (Note, however, that the preset’s name is changed only if you click the “Change Preset Name” button.) See Figure 7.

When you’re done editing the sliders, press Done! to exit the screen, or double-click another preset to edit.

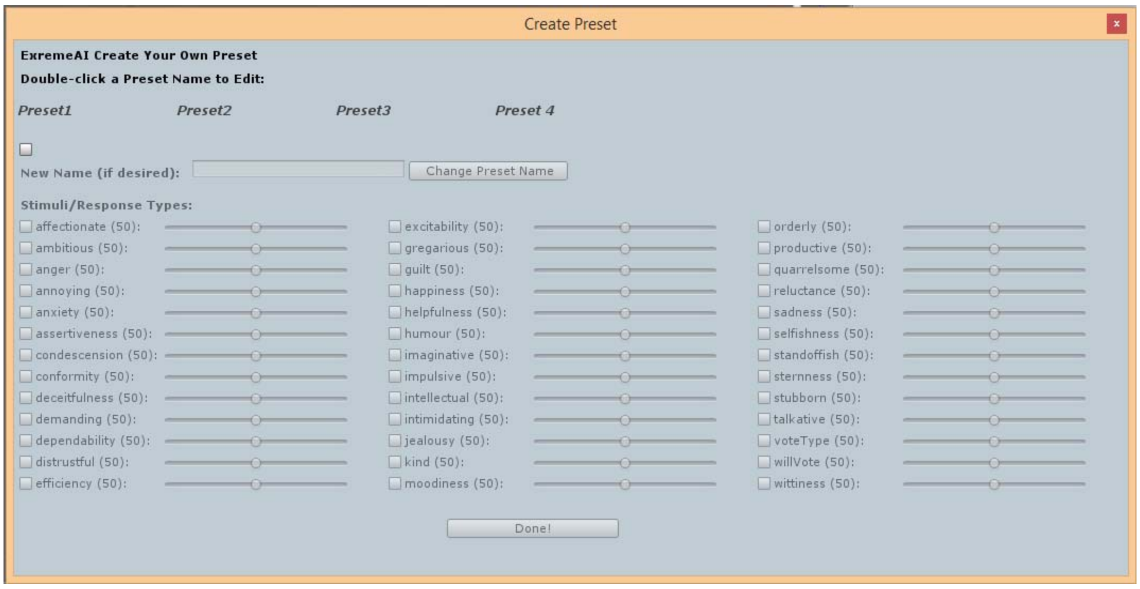
You can now use your new preset to create NPCs!

2.1.6 Other ExtremeAI menu functions

Register Player

As mentioned before, in order for your NPCs to keep track of their attitudes and feelings toward your player(s), they have to know how to reference him/her/them. in this window you can type in the name of a player and press “Register Player”. That’s it! Note that you’ll receive a message telling you whether the player was registered successfully or already exists.

Remember that the player name you register is the one that must be referenced when checking/changing NPCs’ atttudes toward that player, even if you have options in game


Create Preset

ExremeAI Create Your Own Preset

Double-click a Preset Name to Edit:

Preset1 *Preset2* *Preset3* *Preset 4*

☐

New Name (if desired): Change Preset Name

Stimuli/Response Types:

☐ affectionate (50):	☐ excitability (50):	☐ orderly (50):
☐ ambitious (50):	☐ gregarious (50):	☐ productive (50):
☐ anger (50):	☐ guilt (50):	☐ quarrelsome (50):
☐ annoying (50):	☐ happiness (50):	☐ reluctance (50):
☐ anxiety (50):	☐ helpfulness (50):	☐ sadness (50):
☐ assertiveness (50):	☐ humour (50):	☐ selfishness (50):
☐ condescension (50):	☐ imaginative (50):	☐ standoffish (50):
☐ conformity (50):	☐ impulsive (50):	☐ sternness (50):
☐ deceitfulness (50):	☐ intellectual (50):	☐ stubborn (50):
☐ demanding (50):	☐ intimidating (50):	☐ talkative (50):
☐ dependability (50):	☐ jealousy (50):	☐ voteType (50):
☐ distrustful (50):	☐ kind (50):	☐ willVote (50):
☐ efficiency (50):	☐ moodiness (50):	☐ wittiness (50):

Done!

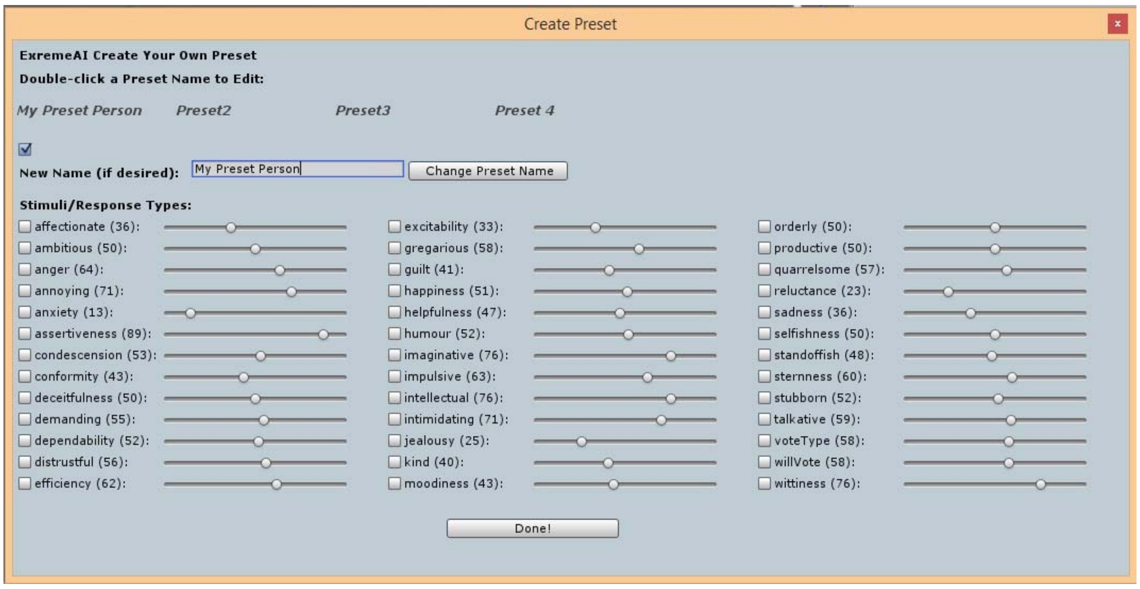
Figure 6 Double-click a preset name to edit

for the human playing the player character to change names. For instance, if you've registered "Player1" as the player name, and in-game that name is changed to Joe the Barbarian, references in the personality engine would still be to "Player1".

Note that you can also set up non-player "players" with which to interact. For instance, you can create a way to have the NPC react to world events by creating a "world_events" player.

The generic name "Player" has been pre-registered for you. Also, any characters you create using the Character Editor or presets will automatically be registered for you, so that they can interact with each other as well.

New in version 2: You can see a list of currently registered players in the drop-down list next to the label "Player List." Selecting one of these does not do anything; the list is there for reference only.


Create Preset

ExremeAI Create Your Own Preset

Double-click a Preset Name to Edit:

My Preset Person *Preset2* *Preset3* *Preset 4*

☑

New Name (if desired): My Preset Person Change Preset Name

Stimuli/Response Types:

☐ affectionate (36):	☐ excitability (33):	☐ orderly (50):
☐ ambitious (50):	☐ gregarious (58):	☐ productive (50):
☐ anger (64):	☐ guilt (41):	☐ quarrelsome (57):
☐ annoying (71):	☐ happiness (51):	☐ reluctance (23):
☐ anxiety (13):	☐ helpfulness (47):	☐ sadness (36):
☐ assertiveness (89):	☐ humour (52):	☐ selfishness (50):
☐ condescension (53):	☐ imaginative (76):	☐ standoffish (48):
☐ conformity (43):	☐ impulsive (63):	☐ sternness (60):
☐ deceitfulness (50):	☐ intellectual (76):	☐ stubborn (52):
☐ demanding (55):	☐ intimidating (71):	☐ talkative (59):
☐ dependability (52):	☐ jealousy (25):	☐ voteType (58):
☐ distrustful (56):	☐ kind (40):	☐ willVote (58):
☐ efficiency (62):	☐ moodiness (43):	☐ wittiness (76):

Done!

Figure 7 Change the name and adjust the sliders

About Quantum Tiger Games
Self-explanatory, really. Clicking this on the ExtremeAI menu gives you more info about the personality engine, including the version number, and Quantum Tiger Games.

Updates and Registration
Also self-explanatory. You can register your copy of Extreme AI, giving you quick access to updates and other information. Also talks about us advertising any games you create using Extreme AI and (hopefully!) giving us a credit in your game.

Personality Info Tool
(Full version only.) Gives you the ability to see the underlying 30 facet values for a character, rather than just the stimulus/response values shown in the character editor. You type a character's name in the box, click the Check Character button, and the character's facet values will print in the Debug Log. Also displays any specific player adjustments for this character (typically none for newly created characters). Click Done to exit the window.

2.2 Using the character personalities in a game
Now that you have the base personalities for your NPCs, it's time to make them game-accessible. Make sure the TaiPE_Lib_v2 or TaiPE_Lib_Light_v2 dll has ben added to your Scripts folder, as per the installation instructions (see Section 1.2).

2.2.1 For each NPC …
For each NPC using the personality engine, attach a script creating its own instance of AICharInfo (the interface between the character and the engine). It also must know how to use the TaiPE_Lib_v2.dll. For example, in C# you would:

1) place a "using TaiPE_Lib_v2;" statement before the class definition in your code ("using TaiPE_Lib_Light_v2;" in the Light version)
2) declare a variable of type AICharInfo (e.g., "AICharInfo myAICI;")
3) instantiate a copy of the module (e.g., "myAICI = new AICharInfo();")

Please see the example Unity code in Part 3.

2.2.2 Naming the NPC
You should also set a variable equal to the NPC's name (matching a character whose personality you created when creating characters using the ExtremeAI menus). The NPC's name will be needed when you query the engine on his/her behalf. Note that this connection is all you need to associate personalities with NPCs.

For example, if you've created a personality for Edgar the Excellent in the database, then in scripts accessing Edgar's personality you'll need a string variable set to "Edgar the Excellent". The engine is case-sensitive.

2.2.3 What you can do
TaiPE_Lib_v2 (or TaiPE_Lib_Light_v2) contains the following user-accessible functions:

Init(string myTag, string myNameIs, bool initFromEditor = true, string savePath = "") [returns nothing]
ChangeBaseChange(float amount) [returns nothing]
ChangeBkgrndChange(float amount) [returns nothing]
AIReturnResult(string playerName, string stimulus, bool toChange = true, bool posChange = true) [returns int]
AINoResult(string playerName, string stimulus, bool posChange = true) [returns nothing]
SavePersonality() [returns nothing]
SavePersonalityAndInit() [returns nothing]
CreateCharacter(string characterName, float[] facetArray, bool usePreset = false, string presetName = "") [returns string]
DeleteCharacter(string characterName) [returns int]
GetCharNames() [returns array of strings]
GetPlayerNames() [returns array of strings]

Advanced functions (available only in the full version of Extreme AI) include:
CharAttValueTool(ref int[] atrArray, ref string[] nmArray) [returns nothing]
CharAttSingleValueTool(string attributeName) [returns int]
CharFacetsTool(ref string[] charFacetNames, ref float[] charFacets) [returns nothing]
CharSingleFacetTool(string facetName) [returns float]
CharVsPlayerValuesTool(ref string[] playerNames, ref string[] facetNames, ref float[] facetValues) [returns int]
CharVsOnePlayerValTool(string playerName, string facetName) [returns float]
ChangeSingleFacetTool(string facetName, float facetValue) [returns nothing]

These are described in detail below (and again, see the examples in Part 3). Note that facet names begin with a capital letter (e.g., Warmth), while stimulus/response/attribute names are all lower case (e.g., kind).

Init(string myTag, string myNameIs, bool initFromEditor = true, string savePath = "") [returns nothing]
This function initialises the AI module, and must be called before you try to use it. “myTag” is the Unity Tag value of this character in the Inspector (so that the GameObject that is the NPC could be located by, for example, GameObject.FindWithTag(myTag). “myNameIs” is the name that matches the NPC’s entry in the personality engine.

New in version 2: “initFromEditor” allows you to make ExAI act as though you aren’t in the Unity editor when you run code. If you are in the editor, the default is to load your personalities from the Resources folder–but saves are to the savePath (either the default savePath or your own–see below). Thus, every time you run your game from within the editor, your savePath is overwritten. (This is also true if you run a new Init from a character later in the same run while in the editor–it will reload the Resources. This does NOT happen when actually running a game outside the editor.) Now you can tell the editor not to do this by setting initFromEditor to false. Note that initFromEditor has no effect when running a game outside the Unity editor (as you aren't in the editor in the first place).

New in version 2: Also new, "savePath" is the directory path you wish to use to save the character personality when saving; leaving this out of the Init means ExAI will use the default save path (which is Application.persistentDataPath + "/ExtremeAI/Resources/Tables"). Use "savePath" with caution; you must have permission to write to the directory to which you're trying to save.

If you are in the Unity editor, ExAI always loads the character values from Resources (that is, the values as you've created them in the editor) into the savePath directory; it does not overwrite the personalities in your Resources folder, no matter what you do in code while running your game. If you aren't in the Unity editor, Init will load from Resources the first time the game is run (assuming the savePath directory doesn't exist), but will thereafter load from the savePath directory (unless it's deleted, or you change the savePath directory without creating it first; see examples in Part 3).

ChangeBaseChange(float amount) [returns nothing]
This function changes the base rate at which the NPC's personality changes in relation to an individual player (or event type). The default is 5.0. This is a bit more quickly than people's personalities change in real life, but to notice the changes in a game, wherein the player may only speak to an NPC a few times, this is the value chosen after testing various rates. Remember that changing this rate for one NPC does not change it for any other (so you can have characters whose personalities differ in terms of rate of change, just like humans).

ChangeBkgrndChange(float amount) [returns nothing]
This function changes the base rate at which an NPC's personality changes in relation to EVERYTHING; i.e., if one player is particularly nice to the NPC, this rate determines the amount the NPC is more kindly disposed to everyone in the world as a result. The default for this rate is 0.1, which again is higher than the rate for humans in the real world, but is desirable for noticing the changes in-game. Changing this rate by even a fraction has a large effect; raising it as high as 1 would create a wildly changing personality.

Again, changing this rate for one NPC does not change it for any other.

AIReturnResult(string playerName, string stimulus, bool toChange = true, bool posChange = true) [returns int]
This is one of the primary functions for the Personlaity Engine; it takes a stimulus/response type (e.g., kind) and returns an integer from 0 to 4 indicating the intensity of the response (see top of next page).

Why only five possible intensities? Because we had to choose between further shades of intensity and speedier response times from the engine. Five seemed the best compromise.

AIReturnResult can be used in different ways. The default takes the character name,

0	Extremely low; could indicate a reaction opposite to that queried, or just an absolutely neutral response (e.g., a kindness intensity of 0 could mean the NPC just ignores the player, or perhaps reacts negatively—the actual reaction is that which suits the situation. Or, if another check indicates the NPC has a high anger intensity, perhaps the NPC reacts very negatively to the player's kindness … it's up to you how many checks you want to make for any given interaction)
1	Low; slightly negative response, or relatively neutral
2	Average; reacts as the "average" person would in this situation
3	High; reacts positively (note that having a high, positive reaction for one stimulus (e.g., kindness) may be very different from a high, positive response to another (e.g., anger)
4	Extremely high; intensely positive response

player name, and stimulus/response type, and changes the personality of the NPC in regard to both this player and to everything else in the world (see also the discussion of the base change functions, above). Alternatively, you can call the function with an added boolean indicating that no changes are to be made to the personality (done by setting the toChange bool to false). You may want to do this when you are simply checking on an NPC's reaction intensity but aren't in a situation where the personality would be changed, such as checking whether an NPC would be the first to start a conversation (checking gregariousness); no real interaction with a player has yet occurred, so there isn't a reason to change the NPC's reactions. There also may be times when you don't know at the outset in what way the NPC's personality would be changed by an event; say you check her impulsiveness to see whether she'll follow the player off the edge of a cliff. It would depend on the outcome of this action whether she later thought of it as a negative or positive reinforcer of her impusiveness. Which brings us to …

AINoResult(string playerName, string stimulus, bool posChange = true) [returns nothing]
So your NPC has jumped over a cliff and had an absolutely smashing time … in the best sense of the word. Her impulsiveness should be increased, but you don't need any sort of value returned for a new response—the response and event already happened.

That's when you use AINoResult. This function changes the NPC's personality without returning a result. The default, taking the character name, player name, and stimulus, changes the values in a positive way, indicating a step towards more intenisty the next time the stimulus is checked.

What if our NPC had had a smashing time in the worst possible sense of the word? You could then change her impulsiveness to be less likely to rule her thoughts by calling AINoResult with an additional boolean value (posChange) set to false.

SavePersonalityAndInit() and **SavePersonality()** [returns nothing]
While NPCs' personalities change in-game with no need to save them, in order to keep changes after a player quits or closes (or reaches a save point) you will need to call

SavePersonalityAndInit or **SavePersonality** for each character using the engine. Because it can take a noticeable amount of time, we'd suggest only doing this at save points or other instances when the gameplay isn't impacted.

Use **SavePersonalityAndInit** in any save situation. It saves the personality (which encrypts the xml files) and then reinitializes it (necessary if play is going to continue, as the files have to be decrypted again). If quitting the game entirely, you can use **SavePersonality**, which doesn't reinitialize.

Note that to be able to return to earlier instances of NPCs' personality states, you will need to save a copy of all tables in the Application.persistentDataPath + "/ExtremeAI/Resources/Tables" folder (or whatever you've set the savePath variable to), which is where files are saved during play (save your copies somewhere else, obviously). When reverting to a saved instance, you'll need to move the copied tables back into the Application.persistentDataPath + "/ExtremeAI/Resources/Tables" folder, overwriting the tables that are already there.

New methods in version 2:
CreateCharacter(string characterName, float[] facetArray, bool usePreset = false, string presetName = "") [returns string]
CreateCharacter allows you to create character personalities on the fly, in code, in-game. It does not adjust facets to make them more "realistic" in relation to one another (as in done in the Character Editor in Unity); whatever you set the facets to, that's what they are. NOTE: In the Character Editor, you are adjusting the underlying facets by making changes to the stimulus/response types. In the CreateCharacter method, you are creating facets directly. To give you an idea of how the facets relate to the stimulus/response types, we've added an Appendix to this manual.

To use **CreateCharacter**, send:
characterName–a string representing the name of the character, which you'll use to access his/her personality. If the character name already exists, the method will fail.
facetArray–an array of 30 facet values, in the order shown in Table 1. Facet values range from 0.0 to 100.0.
usePreset–(optional) set to "true" if, instead of using your own facet values, you wish to use a preset (one of those in the Choose Preset menu). Note that you still have to send something in the variable facetArray for the method to work (even though facetArray won't be used); however, you don't have to send 30 values, just some kind of dummy float array.
presetName–(don't use unless you set usePreset to "true") a string representing the name of the preset you wish to use to create the character. This string must match one of the presets in the Choose Preset menu, or the method will fail.

The returned string will indicate whether you successfully created a character, as follows:
"success"–the character was created successfully
"failCharExists"–the character already exists, and thus was not re-created
"failIncorrectFacetNumber"–the wrong number of facets was sent (must be 30)
"failNoSuchPreset"–a non-existent preset was used

DeleteCharacter(string characterName) [returns int]
This method allows you to delete any character. It is deleted completely (although not from the original Resources data used to create a new game); there is no "undelete." You simply send the character name, and that character is gone. This method returns the character ID if successful, and 0 if it fails. (It will fail if the character doesn't exist in the first place.)

GetCharNames() [returns array of strings]
Want to know the names of all your characters? This method returns an array of all character personality names.

GetPlayerNames() [returns array of strings]
Similarly, if you want to know the name of every registered player (which should include all the character names, plus those you've registered as players only [including "Player" and anything else you've entered in the Register Player menu in the editor]), use this method. It returns an array of all player names.

Advanced methods available in full version only:
CharAttValueTool(ref int[] atrArray, ref string[] nmArray) [returns int]
This tool allows you to get an array (by reference) of all 39 raw stimulus/response attributes for a character. Actually, you get two arrays: one is the attribute values, one is the attribute names. Probably most useful in testing a game, so you can get a snapshot of how a character is changing. Takes a noticeable amount of time to return, so not a good idea to run this in the middle of a lot of game action. Note that these are raw values, not calculated personality-based responses (for which you'd use **AIReturnResult** or **AINoResult**).

CharAttSingleValueTool(string attributeName) [returns int]
This tool returns a single raw stimulus/reponse attribute value (as opposed to calculating a character's actual personality-based response; for that you'd use **AIReturnResult** or **AINoResult**). You provide the attribute name, and the method returns the value of that attribute. Note attribute names are lower case (e.g., kind).

CharFacetsTool(ref string[] charFacetNames, ref float[] charFacets) [returns nothing]
This tool allows you to get an array (by reference) of all 30 raw facet values for a character (you also get a matching array of the facet names). This shows you the character's personality at its most basic level. Again, this doesn't provide a calculated personality-based response (for which you'd use **AIReturnResult** or **AINoResult**).

CharSingleFacetTool(string facetName) [returns float]
This tool returns a single facet value (any of the 30 underlying facets in the Five-Factor model), and thus a snapshot of the most basic level of a character's personality. Again, this doesn't provide a calculated personality-based response (for which you'd use **AIReturnResult** or **AINoResult**). Note facet names begin with a capital (e.g., Warmth).

CharVsPlayerValuesTool(ref string[] playerNames, ref string[] facetNames, ref

float[] facetValues) [returns int]
Returns (by reference) all player/other NPC/world-based adjustments to a character's underying facets, thus showing you how this character feels about specific individuals. Most useful in testing a game; it can take a noticeable amount of time to return (especially as the game goes on and an NPC has met a lot of other people and/or had a number of interactions with a few people), so not a good idea to run this in the middle of a lot of game action. These are obviously not calculated personality-based responses.

The int returned tells you whether or not any values were returned. If this NPC has not encountered any players that changed her personality, the int will equal 0.

CharVsOnePlayerValTool(string playerName, string facetName) [returns float]
Returns a character's adjustment value for a single facet toward a specific player/other NPC/etc. Let's you know how much more/less a character feels, for instance, trusting of the player than he does a total stranger (or affectionate, or any of the other facets).

ChangeSingleFacetTool(string facetName, float facetValue) [returns nothing]
Changes a single facet of this character. Use a facet name from Table 1 for facetName, and a value between 0.0 and 100.0 for facetValue.

See Part 3 for examples of the basic functions in use, and take a look at the sample Unity scene for a demonstration. You can also visit ***quantumtigergames.com*** for another example (the voters demo).

3.0 Examples

Of course, just the cut-and-dry enumeration of function calls doesn't really demonstrate the possibilities, the range of character responses and development when using the ExtremeAI engine. Following are a few examples of using the engine in a Unity environment. (Note that the examples assume you are using C#.)

3.1 Remember to set the NPC and player names!

As mentioned before, it's a good idea to set a string variable to the name of the NPC whose personality you are accessing, as this name will be necessary every time you query the database. However, you can change an NPC's personality from any script, anywhere, as long as you use his/her name in the function call.

Also, the Engine must know which player (or thing) is interacting with the NPC in order to return values that make sense. Setting the player name is done by registering it in the ExtremeAI menu, as described in section 2.1.6.

3.2 Getting it all started: using Init

To use the engine, you have to create an instance of it for a character and then initialize it. This is easy to do: in a script you attach to a character, do the following (and see the boldface items in Figure 3.1):

1) make sure to include a "using TaiPE_Lib_v2;" (or "using TaiPE_Lib_Light_v2;" in the Light version) statement above the class declaration
2) when declaring variables for use in the class, instantiate a copy of AICharInfo
3) in the Start function provided by Unity (or some other function called when the script is first called), continue the instantiation (C# requires not only declaring it in the variables, but then setting it equal to a "new AICharInfo()")
4) also in the Start function, call the Init(myTag, myNameIs) function of AICharInfo to initialize the Engine for this character

That's it! Now you're ready to access the character's personality, which you created using the character setup menus described previously (see Section 2.1).

If you're in the Unity editor, you can use initFromEditor to choose whether to load all your personalities from the Resources folder or from elsewhere. See **Init** on page 12 for details.

If you wish to use a different save path than the default (Application.persistentDataPath + "/ExtremeAI/Resources/Tables"), let ExAI know what it is. For example:

```
string mySavePath = Application.persistentDataPath + "/myCoolGame/CharacterSaves";
myAICI.Init(this.tag, NPCName, true, mySavePath);
```

You should make sure you have permission to write to whatever directory you choose (for example, there are folders that Windows won't allow you to write to). Note that if the directory already exists, ExAI will try to load game data from it. If the directory does not exist, ExAI will load from your game's original Resources and save immedi-

```
using UnityEngine;
using System.Collections;
using TaiPE_Lib_v2;

public class NPCClientSide : MonoBehaviour {

        /*
                Author: Jeffrey Georgeson, build 12 May 2017
                Copyright Quantum Tiger Games, LLC 2017
        */

        //who is the NPC we're dealing with in this script?
        string NPCName = "Edgar the Excellent";

        //instantiates copy of AICharInfo
        AICharInfo myAICI;
        //alternatively, could use AICharInfo myAICI = new AICharInfo();
        //which would take care of instantiation all at once

        ...

// Use this for initialization
        void Start () {

                //continue instantiation of AI interface
                myAICI = new AICharInfo(); //unless already done through alternate
                //code above

                myAICI.Init(this.tag, NPCName);

                ...
```

Figure 3.1 Initializing a copy of AICharInfo for a character

ately to your new directory, creating it in the process.

3.3 Changing speed of change

If you want to change the speed at which this character's personality changes (not recommended until after you've used the PE for a while to get a sense of how it works), now would be the time to do it—just after initializing the Engine, still within the Start function. (Not that this is required. The speed at which different facets change in real people's lives is already factored into our Engine, and so continually tinkering with the overall speed of change will make the NPC more unrealistic and possibility quite wacky.)

To do this (see Figure 3.2):
1) call the **ChangeBaseChange** function of your instance of the AIClientInterface (in the example we've called it "myAICI"), with a float representing the new value. This will change per interaction the attitude the NPC has toward specific individuals, rather than change the overall personality. The default is 5. Note that whole number changes in this value will have a very significant effect.

```
//use the following function to change the rate of change per interaction with each player.
//this represents the attitude the NPC has toward specific individuals, rather than the overall
//change (i.e., towards Mike the First Player, but not towards everyone and everything
//generally).
//note that the default is 5f

float someAmount = 5f;
myAICI.ChangeBaseChange(someAmount);


//use the following function to change the rate of change for every interaction the NPC has.
//this is like changing the rate an overall personality changes. In the real world, such changes
//usually take years; the default rate for gameplay is 0.1f (which is still faster than real-world
//changes, but necessary for most gameplay situations).

float someOtherAmount = 0.1f;
myAICI.ChangeBkgrndChange (someOtherAmount);
```

Figure 3.2 Altering the speed at which a character’s personality changes

```
private bool StartConversation()
{

    //determine whether NPC wishes to start the conversation, or wait for the player to
    //start. Use the AI to determine by checking against NPC's gregariousness

    //pass NPC name, player name, tendency to check, and whether to change
    //personality based on this check (in this case, we don't; just getting a result)

    int startConversation = myAICI.AIReturnResult(plName, "gregarious", false);

    //this returns a value between 0 and 4, 0 being low in this tendency, 4 being very high

    //we don’t need a separate response for every return value, so ...
    if (startConversation > 2)
    {
        //will definitely start conversation
        return true;
    } else if (startConversation < 2) {
        //will definitely not
        return false;
    } else {
        //in between; in this case, decided randomly
        System.Random random = new System.Random();
        int randomNumber = random.Next(0, 100);
        if (randomNumber >= 50)
        {
            return true;
        } else {
            return false;
        }
    }
}
```

Figure 3.3 Getting a result without changing the NPC personality

2) call the **ChangeBkgrndChange** function of myAICI to change the rate of change for every interaction the NPC has. This is like changing the rate an overall personality changes. In the real world, such changes usually take years; the default rate for game-play is 0.1 (which is still faster than real-world changes, but necessary for most game-play situations).

3.4 Using AIReturnResult and AINoResult

Sometimes you will want to get a result from the NPC's personality, but not have the mere fact of getting a result alter that personality. In the example in Figure 3.3, we are checking whether an NPC will start a conversation with a player who has just entered the NPC's shop. We don't want to change the NPC's personality (maybe we'd change it after the encounter, though, if for instance things go well and reinforce the NPC's desire to start conversations), so we check using **AIReturnResult**, but with the toChange parameter set to false.

To change the values later without needing another result, we could use **AINoResult**. For example, assume that after the conversation, we want to change the personality depending on whether we've evaluated things to have gone well (and thus reinforcing a stimulus, in this case the desire to start another conversation) or badly (thus making it less desirable to start conversations in the future, especially with this player). We would write something like the code in Figure 3.4, adjusting in a positive direction (the default) or a negative one.

The default action for any personality check requiring a result is to adjust the NPC's personality at the same time (as this seems the more likely pattern). Thus a player being intimidating to an NPC will immediately change the NPC's personality; there's no need to wait until later to evaluate things. In this case, call AIReturnResult with toChange and posChange set to True (the default, so really you need only have the playerName and stimulus parameters) or with toChange set to True and posChange set to False (see Figure 3.5).

```
private void AfterConversation(bool thingsWentWell)
{

    //after the conversation, we decide to adjust the NPC's gregariousness based on
    //whether there was a positive outcome or a negative one

    //pass NPC name, player name, tendency to check, and whether to change
    //personality in a positive direction or a negative one

    if(thingsWentWell)
    {
          AINoResult(plName, “gregarious”);
          //note that the default is for positive change
    } else {
          AINoResult(plName, “gregarious”, false);
    }

}
```

Figure 3.4 Changing the personality without returning a result

```
//player has been kind
        whichResponse = myAICI.AIReturnResult(plName, "kind", true, false);
        switch (whichResponse)
        {
                case 0:
                        NPCResponse = "<replies something negative>";
                        break;
                case 1:
                        NPCResponse = "<replies something neutral>";
                        break;
                case 2:
                        NPCResponse = "<replies something kind>";
                        break;
                case 3:
                        NPCResponse = "<replies something really kind>";
                        break;
                case 4:
                        NPCResponse = "<offers help to player>";
        break;
        }

        ...
```

Figure 3.5 Changing the personality (negatively) and returning a result

In the above example, the player is kind to the NPC, and so she checks her response to kindness and changes slightly for the experience. There are more complex ways to check (and change) the NPC's personality, however. In Figure 3.6, we see an example of an NPC being intimidated by a player. In this case, we decide to check not just one but two response types—one for the NPC's intimidation response, and one for her anxiety response. Why? Because her desire to be intimidating right back might be outweighed by her anxiety over the situation. So in this case we check both, evaluate them against each other, and then figure out the NPC's response.

```
//so we're checking how the NPC responds to intimidation (is she more
//anxious or intimidating right back?) using a more complex combination
//of responses

        int isAnxious = myAICI.AIReturnResult(NPCname, plName, "anxiety");
        int isIntimidating = myAICI.AIReturnResult(NPCname, plName, "intimidating");

        whichResponse = isIntimidating - isAnxious;

        switch (whichResponse)
        {
                case -4:
                        NPCResponse = "<calls for help!>";
                        break;
                case -3:
                        NPCResponse = "<replies something frightened>";
                        break;
                case -2:
                        NPCResponse = "<replies something nervous>";
                        break;
                case -1:
                        NPCResponse = "<replies something neutral>";
                        break;
                case 0:
                        NPCResponse = "<replies something neutral>";
                        break;
                case 1:
                        NPCResponse = "<replies something neutral>";
                        break;
                case 2:
                        NPCResponse = "<replies something annoyed>";
                        break;
                case 3:
                        NPCResponse = "<tells player to get lost>";
                        break;
                case 4:
                        NPCResponse = "<ATTACKS!>";
                        break;
        }
```

Figure 3.6 More complex use of AIReturnResult

4.0 Support and Credits

4.1 Support

We’re here to help! Contact us at support@quantumtigergames.com with any issues. We’ll also post a FAQ on our website (once we have enough questions to have a FAQ!).

4.2 Credits

Design, coding: Jeffrey Georgeson
Original Raina character created by Mark Foley. Copyright © and TM 2012 Quantum Tiger Games, LLC

RealMemory: We have an even more specific memory package for NPCs, in which they can remember actual events and the people attached to them, rather than just reaction types. See RealMemory on our website, and in the Unity Asset Store! (You’ll get a special upgrade deal if you already own the personality engine!)

5.0 Citations

What!? References in a user manual? Yes, indeed; because our game AI is based on research dealing with real humans and personality development. The following are a few of the many works dealing with the Five Factor Model of personality:

Costa, PT, Jr and McCrae, RR 1995, ‘Domains and Facets: Hierarchical Personality Assessment Using the Revised NEO Personality Inventory’, *Journal of Personality Assessment*, vol. 64, no. 1, pp. 21-50.

DeYoung, CG 2010, ‘Toward a Theory of the Big Five’, *Psychological Inquiry*, vol. 21, no. 1, pp. 26-33.

Goldberg, L 1993, ‘The structure of phenotypic personality traits’, *American Psychologist*, vol. 48, no. 1, pp. 26-34.

John, OP et al 2010, ‘Paradigm Shift to the Integrative Big Five Trait Taxonomy: History, Measurement, and Conceptual Issues’, in John et al (ed.), Handbook of personality: Theory and research, 3d ed, Guilford, New York. (US edition.)

McCrae, RR & Costa, Jr PT 2010, ‘The Five-Factor Theory of Personality’, in John et al (ed.), *Handbook of personality: Theory and research*, 3d ed, Guilford, New York. (US edition.)

Appendix
How Facets Relate to Response Types (Generally)

In the new CreateCharacter method, you can create new characters on the fly, in-game. However, rather than adjusting stimulus/response types to change character facet values (as is done in the Character Editor in Unity), CreateCharacter lets you input the facet values directly. This is mostly because it would be complicated and resource-intensive to try to implement something like the Character Editor within a game, causing very noticeable slowdowns for every change in a stimulus/response value (it causes a pause even in the Editor).

So that you can create characters with some idea of how the facets relate to the response types, we have provided the table on the following pages. mod = moderate; (n) = negative correlation

Stimulus/ Response	**Facets Involved**	**Weighting**
affectionate	warmth feelings	high mod
ambitious	achievement	high
anger	angry hostility self-consc warmth assertiveness positive emo trust compliance	high mod high (n) high mod (n) mod (n) mod (n)
annoying	warmth anxiety self-consc assertiveness angry hostility straightfwd gregarious	high (n) mod (n) mod high high mod mod (n)
anxiety	anxiety depression vulnerability assertiveness	high mod mod high (n)
assertiveness	vulnerability assertiveness activity	mod (n) high mod
condescension	gregarious modesty	mod (n) high (n)
conformity	feelings values compliance dutifulness assertiveness	mod (n) mod high high mod (n)
decitfulness	straightforward dutifulness	high (n) mod (n)
demanding	angry hostility modesty order	high mod (n) mod
dependability	competence dutifulness achievement self-disc fantasy vulnerability	high high mod high mod (n) mod (n)
distrustful	self-consc trust altruism tender-minded	mod high (n) mod (n) mod (n)
efficiency	assertiveness competence achievement	mod mod high
excitability	anxiety activity deliberation self-disc	high mod (n) mod (n) mod (n)
gregarious	warmth anxiety self-consc assertiveness depression positive emo gregariousness	high mod (n) mod-high (n) mod-high mod (n) mod high
guilt	anxiety depression self-consc	mod mod mod
happiness	angry hostility gregarious positive emo feelings trust	mod (n) high mod high mod
helpfulness	altruism tender-minded trust	high mod mod
humour	fantasy aesthetics warmth	mod mod high
imaginative	fantasy	high
impulsive	assertiveness activity order dutifuleness self-disc deliberation	high high mod (n) high (n) mod (n) mod (n)
intellectual	ideas values	high mod

Stimulus/ Response	Facets Involved	Weighting
intimidating	warmth anxiety self-consc depression angry hostility vulnerability gregarious assertiveness trust compliance	mod (n) high (n) mod high (n) high mod (n) mod (n) high mod (n) mod (n)
jealousy	vulnerability modesty	high high (n)
kind	warmth feelings trust altruism angry hostility	high mod mod high high (n)
moodiness	angry hostility depression activity positive emo feelings	high mod mod (n) mod (n) mod (n)
orderly	competence order achievement self-disc	mod high mod high
productive	competence order dutifulness achievement self-disc	mod mod mod high high
quarrelsome	angry hostility impulsive gregarious warmth tender-minded	high mod mod (n) mod (n) high (n)
reluctance	assertiveness anxiety self-consc impulsiveness vulnerability	mod (n) high mod mod (n) mod

Stimulus/ Response	Facets Involved	Weighting
sadness	depression impulsive vulnerability activity positive emo feelings	high high mod mod (n) mod (n) mod (n)
selfishness	altruism tender-minded	high (n) mod (n)
standoffish	warmth gregarious positive emo trust altruism tender-minded	mod (n) mod (n) mod (n) mod (n) mod (n) mod (n)
sternness	gregarious	mod (n)
stubborn	vulnerability assertiveness compliance achievement	mod mod mod (n) mod
talkative	warmth gregarious assertiveness positive emo deliberation	mod high mod mod mod (n)
voteType	values anxiety ideas aesthetics fantasy	high mod (n) mod mod mod
willVote	assertiveness positive emo	high high
wittiness	fantasy ideas	high high