Following is a list of the CODE Files for the AGAI Project:

1. AIFollow.js
2. AIfollowEnemyAvider.js
3. AIfollowsmallenemy.js
4. Collector1.js
5. Collector2.js
6. Collector3.js
7. Collector4.js
8. Cube1AI.js
9. Cubesmall1AI.js
10. CUBESMALLAI.js
11. DamagePotionScript.js
12. GoodHealthScript.js
13. Ground.js
14. INITIAL.js
15. NPC.js
16. Talker.js

Out of all these files the main file related to our AI Implementation is the file NPC.js. This file contains all the code for the implementation of ALMA (A Layered Model of Affect) paper. The file contains classes for Personality (Big Five), Emotions and Mood as required by the ALMA paper. Then we create a NPC class that can then link all of these classes using the push pull method. Detailed description of the code follows:

Since, we are using Unity's version of JavaScript, we can define regular classes in it. The first class we define in the file NPC.js is Personality class. The class contains five double variables for the BIG FIVE characteristics: openness, conscientiousness, extraversion, agreeableness and neuroticism. We also define the parameterized and the default constructor for this class.

Then, we define the Emotion and the Mood classes, which are the crux of ALMA implementation. Both the classes contain three doubles for the Pleasure, Arousal and Dominance values. We define the parameterized and the default constructors for these classes.

Then, we define the main class Player. It contains a Personality variable, an Arraylist of Emotions called Active Emotions and a Mood variable representing its current PAD value. Its parameterized constructor takes in five doubles for the personality. We then use the three equations mentioned in the ALMA paper to calculate the default Mood (PAD values) for the character. Its active emotions array list is currently empty because we do not have any ACTIVE EMOTIONS.

We define 4 MAIN FUNCIONS:

1. pushPull(): In this function, we return straight back, if the emotion list is empty, because we do not perform push-pull if they are no active emotions. If there are active emotions, we first compute the emotion center of these emotions. Then we check if the emotion center and the current mood are in the same quadrant. If they are not in the same quadrant, then we call pull(). If they are in the same Quadrant we call sameQuadrant() function.

2. sameQuadrant(emotioncenter): In this function we first find out the quadrant in which the Emotion Center and the Current Mood lie. Determining this quadrant is important for checking if the currentmood is between the zero of the PAD space and current virtual emotion center. Then we decide whether to push or pull

3. pull(emotioncenter):

In pull function, we pull the PAD values towards the current Emotion Center. We calculate the difference between the mood PAD values and those of the Emotion Center. We then multiply this change values by the magnitude for the change. This reduces the change values to make sure the changes are not very sudden. We then multiply the change value by the intensity of the virtual emotional center, so that changes are proportional to the average intensity of all the emotions i.e. the higher the intensity the more drastic the change in values.

4. push(emotioncenter):

We use the PAD values of the emotion center to decide by what amount to push. We divide these values by the emotional intensity and multiply by some arbitrary step size (used as 5 in game) to get the change value and add the value to the current Mood PAD values.

No, we maintain a global Hash table of Emotions that map from emotion names (string) to emotion objects containing the PAD values.

In game, when any action is taken by the user and appraised by the player, we fire Emotions using their names. We get their corresponding PAD values from the Hash table. Append all these active emotions to a list and then update the emotion list of the player with this list and call pushPull. Then our pushPull function will change the PAD values.

All changes to behavior and the environment are communicated through static variables which can be found in the code.

We have various Points in the Game where we appraise, fire emotions and call push pull using those emotions. Some of them are:

1. Giving the Good or the Bad potion to the NPC

2. Taking the NPC's side by not giving or giving the scroll to the old man.

3. When the NPC sees his dead friends Corpses

4. When he gives us advice and we decide to heed it or not heed it

Based on the above conditions, we end up with a set of final scenarios that are dependent on the PAD Values:

1. The NPC becomes completely hostile and volunteers to go ahead and destroy the level boss himself, to let us out. (This was originally the NPC becoming hostile towards the player. However, this was found to be rather simplistic and was hence scrapped for the other idea.)

2. The NPC becomes Exuberant, in which case he shows us an alternate path out of the game.

3. If any combination of actions are chosen that do not constitute a final Hostile or Exuberant PAD value, then the NPC still remains the same as before and the level progresses in the normal way.

We thus have a total branching of over 16. However, we have pruned off paths in the course of the game to make the handling of conditions easier. The final Exuberant/Hostile condition itself removes 8 branches from the game by turning them all into the NORMAL GAME conditions.