AI Personality Traits in Freeciv

Project for course AI for Games

Łukasz Faber
Tomasz Miasko
{faber,miasko}@student.agh.edu.pl
Contents

1 Introduction 3

2 Implementation of traits 4
   2.1 Definition of traits ........................................... 4
   2.2 Traits values ................................................. 4
   2.3 Functions ..................................................... 5
   2.4 AI trait command and saves .................................. 5

3 Traits 7
   3.1 Imperialist ................................................... 7
   3.2 Diplomat ...................................................... 10
   3.3 Expansionist .................................................. 10
   3.4 Scientific ..................................................... 14

4 Summary 25
1 Introduction

The main purpose of this project was to extend existing artificial intelligence in Freeciv, a popular open source clone of Civilization game created by Sid Meier. In Freeciv player becomes the leader of a civilization and fight to obtain the ultimate goal: “To become the greatest civilization.”

Goal of this extension was to create a set of personality traits for AI such as imperialist, expansionist or scientific. In such way that each of them could be controlled independently (e.g. between imperialistic and non-imperialistic), and it would be possible to combine them together (e.g. unscientific expansionist).

Together this could lead to more enjoyable game with wide range of possibilities, in which AI could be customised to fit player needs or chosen randomly. Furthermore it can be used to compare and develop even better AI players and strategies.
2 Implementation of traits

The source code that provides a simple framework for specific traits' implementations is located in files ai/traits.c and ai/traits.h. A short description of how it works follows.

2.1 Definition of traits

All traits are defined as enum values in the type ai_trait. They are numbered from 0 to some positive value.

A definition used by us:

```c
enum ai_trait {
    TRAIT_IMPERIALIST = 0,
    TRAIT_CIVILIZED,
    TRAIT_DIPLOMAT,
    TRAIT_IDEALIST,
    TRAIT_EXPANSIONIST,
    TRAIT_SCIENTIFIC,
    TRAIT_LAST
};
```

Moreover, all traits needs to have their name put to an array:

```c
char *ai_traits_names[]
```

This name is used in a description shown in the server command line.

There is one more structure that need to be extended for a trait to work:

```c
trait_callback ai_traits_callbacks[]
```

It contains pointers to functions that need to be called after changing a value of a specific trait for a player. It is correct to put a NULL there instead.

2.2 Traits values

In our implementation every AI player can have every trait set to value of −1, 0 or 1. Its meaning is simple: −1 is a complete negation of a trait (e.g. with “Civilized” trait set to −1 an AI would behave like a real barbarian), 0 is a no-change and 1 is sets a named trait.

Trait values for a player are kept in a structure (located in ai/aidata.h) used by Freeciv for all AI-related data. We just added an additional field:
struct ai_data {
    /* The Wonder City */
    int wonder_city;
    // ...
    int max_num_cities;
    /* AI traits */
    short traits_values[TRAIT_LAST];
};

2.3 Functions

We defined some functions used to operate on traits. These are:

const char* ai_trait_name(enum ai_trait id);
enum ai_trait find_ai_trait_by_name(const char *name);
short player_ai_trait_value(struct player *pplayer,
    enum ai_trait trait);
void set_player_ai_trait_value(struct player *pplayer,
    enum ai_trait trait, short value);

The first one is used to get an AI trait name. The second finds AI trait enum value by a name. The third and fourth are basically a getter and setter of a trait value for a player.

Callbacks

As mentioned earlier, some of traits use callbacks for settings some other AI parameters when their own value is changed. A callback must be a function of a type:

void (*trait_callback)(struct player*, short);

Where the first parameter is a player structure and the second — a new value of this trait.

2.4 AI trait command and saves

To be able to set trait values we added a new command to Freeciv server. The command has a name aitrait and its full syntax is:
aitype <player-name> [trait <value>].

In the full form (with “trait” and “value”) it sets a trait to a given value for a player. In the form with only a player name given it shows values of all traits for this player.

The implementation of this command can be found in a file server/stdinhand.c. Its name is set_ai_trait_command.

A possibility to store traits values in Freeciv save files and to load them also has been added. All values are saved as a simple array using the following code:
for(i = 0; i < TRAIT_LAST; i++) {
    secfile_insert_int(file, ai->traits_values[i],
                       "player%d.ai.trait%d", plrno, i);
}
3 Traits

When analysing results one should remember that traits are very narrow and specific changes to a particular behaviour and sometimes their propagation is not very noticeable in a game.

3.1 Imperialist

Imperialistic AI systematically looks for targets to attack, and stops its warmongering activities only when already at war. On other side we have peaceful AI which would never start a war on its own.

Implementation

Imperialistic behaviour was hitherto typical for AI in Freeciv. Because of this changes concentrated on creating more peaceful AI, which would not provoke wars, and would be more eager to sing peace treaties.

The most basic changes consisted of controlling war threshold and love required for peace. This constants are compared to love that AI has for other players. If love for other player is lower than war threshold and there is “need for war”, AI starts countdown which leads to declaring a war. Second constant is used when evaluating gold equivalence of treaty. Difference between current love and req_love_for_peace i.e. missing love is used to decrease value of treaty.

```plaintext
ai->diplomacy.war_threshold = - MAX_AI_LOVE / (8 + value);
ai->diplomacy.req_love_for_peace = MAX_AI_LOVE / (8 + 2 * value);
```

Another part of changes involved modifying logic used to decide whatever we need target for new war in function ai_diplomacy_actions.

Results

We evaluated imperialist trait in two different test cases. First by comparing results of game where all players had the same value of imperialist trait. In the second case we created game with players having imperialist trait set to -1 and 1 and checked which one would result in better score.

Sample 1

In this test we compare game in which players had imperialistic trait set to −1, to another one in which imperialistic trait was set to 1. In first case the last turn was
on 1706 AD (T290) and one of players have already reached Alpha Centauri. In second case game ended in 1731 AD (T305) with conquest.

Figure 3.1: Example screen-shot from game (peaceful players)

Figure 3.2: Example screen-shot from game (imperialistic players)
Figure 3.3: Comparison of minimaps in penultimate turn (peaceful on the left, imperialistic on the right)

Figure 3.4: Literacy
Sample 2

In the second test we check which value for imperialist will allow to achieve more victories. There were two imperialist players and two non-imperialist. To take into account differences in starting position the game had been run 6 times (all possible assignments of traits to players). Players with imperialist trait set to 1 won 4 times, and players with trait set to -1 won 2 times. So in this case we can see that strategy implemented in Freeciv to this time was better.

3.2 Diplomat

Diplomatic traits controls how likely AI is willing to participate in alliances, to accept treaties and tech exchanges.

This trait was implemented only by modifying existing coefficients used to control change of love between players and in evaluating treaties. Changes were only minor and don’t influenced game results in noticeable way, but allowed us to discover bug in Freeciv implementation – place where integer division was used instead of floating point division, which resulted in statements having no effects, because of rounding to zero.

3.3 Expansionist

An expansionistic AI likes to have a large territory, scattered and small cities. Instead of improving currently existing cities it builds new ones.

On the other hand, a more isolated AI try to build larger cities and focus on improving already existing ones. It avoids having large city count.
Implementation

The version focused on larger cities were quite simple to implement. We created a limit of cities per continent for it and made an AI prefer building improvements than new units (especially settlers). The other side of this trait was little bit harder to create, as a normal version of AI is already very expansionistic (it actually is the best strategy to win the game). In this case we usually just modify "wants":

```c
if (ai->traits_values[TRAIT_EXPANSIONIST] < 0) {
    founder_want /= 10;
} else if(ai->traits_values[TRAIT_EXPANSIONIST] > 0) {
    founder_want *= 20;
}
```

Results

A problem with not-expansionistic AIs is that they work in completely opposite way to the best strategy for Freeciv. Therefore they usually lose. On the other hand results of changes are very noticeable.

Sample 1

In this test “Oliver Cromwell” (red line) had the expansionistic trait set to -1. All other players had no trait changed. In Figure 3.6 we can see “Oliver Cromwell” after the turn 294 — he has very large cities. On the other hand another “normal” player is shown on Figure 3.7 with more smaller cities (and this is common behaviour of AI in Freeciv). Charts showing number of cities and a settled area for players are shown respectively in Figure 3.8 and Figure 3.9. One can clearly notice a limit set on the red player.

![Figure 3.6: Screenshot of a not-expansionistic player](image)
Figure 3.7: Screenshot of a normal player

Figure 3.8: Number of cities for each player
Sample 2

In the second sample we reversed roles — “Oliver Cromwell” had the expansionistic trait set to 1. This time it is clearly visible in the Figure 3.11 that he is very good in land expansion but in some point of time he loses most of it (probably due to a war). Similar pattern can be noticed in the Figure 3.10.
3.4 Scientific

A scientific AI should be focused on technological discoveries and should try to obtain available technologies as fast as possible. It would rather spend money on
research than on military units or luxury.

An unscientific AI should, on the other hand, avoid advancement in technology because it has better ways to spend money.

Implementation

The implementation of this trait is centered on setting a high (or low) value of scientific_want parameter for an AI. All places where this value is used have been replaced my code like this:

```cpp
if(ai->traits_values[TRAIT_SCIENTIFIC] > 0) {
    ai->science_priority = TRAIT_PARAM_SCIENTIFIC_WANT;
} else if(ai->traits_values[TRAIT_SCIENTIFIC] < 0) {
    ai->science_priority = TRAIT_PARAM_UNSCIENTIFIC_WANT;
}
```

Two constants used in this sample code (TRAIT_PARAM_SCIENTIFIC_WANT and TRAIT_PARAM_UNSCIENTIFIC_WANT) are respectively large and small integer values (relatively to other “wants”).

Furthermore, taxes management has been slightly modified. Now, it tries to set a maximum possible rate for science and never allows to cut taxes for science when a player needs money.

Results

A problem with scientific-only AIs is that they usually have worse performance in wars and expansion so they tend to be worse than the rest of players.

Sample 1

In this test “Oliver Cromwell” (red line) had set the scientific trait to 1 and “Mehmûd Berzençî” (green) to −1. One can notice that a more scientific AI is much worse in overall results, like a population size. On the other hand it has more stable technology output value (Figure 3.14) an number of specialists. Better results of a light blue player (“Mary McSwiney”) can be explained by a better initial location.
Figure 3.12: Population

Figure 3.13: Number of techs researched
Figure 3.14: Technology output

Figure 3.15: Science tax
In this test “Oliver Cromwell” (red line) had set the scientific trait to 1 and “Mehmûdê Berzenci” (green) to $-1$. All AIs had set “Imperialist” trait to $-1$ (so they were very peaceful).

This is an example of how well scientific AI can do when surrounded by peaceful people. On the other hand, the green player is very weak because he does only minimal amount of research.
Figure 3.17: Population

Figure 3.18: Number of techs researched
Figure 3.19: Technology output

Figure 3.20: Science tax
Figure 3.21: Number of researchers

Sample 3

In this test “Eudes IV” (purple line) had set the scientific trait to 1 and “Felipe II” (blue) to −1. All AIs had set “Imperialist” trait to −1 (so they were very peaceful). Basically — it is an example 2 with the scientific AI placed in better location. In Figure 3.26 it is noticeable that it has more specialists than other players.
Figure 3.22: Population

Figure 3.23: Number of techs researched
Figure 3.24: Technology output

Figure 3.25: Science tax
Figure 3.26: Number of researchers
4 Summary

We successfully developed framework which introduces traits to Freeciv game. It is extensible way to add new traits to game and modify existing behaviours.

Using it we created traits influencing wide range of AI actions. Letting player to decide if AI should focus on technological discoveries, conquering near civilisations or living with them in eternal peace, building dense network of interconnected cities or small ones scattered all other the world.

We evaluated different strategies allowed by our changes. Results of this evaluation ranged from negligible effects on overall gameplay to easily noticeable differences.

We think it would be worthy addition to Freeciv, and in if extended further could provide even more enjoyable gameplay.

However, it should be noted that Freeciv (especially AI) is very hard to extend. The overall structure of AI is too complicated and too big and stands on a way to make significant changes to its behaviour due to usage of many undocumented constants, variables and parameters. To fully parametrise an AI and create many interesting traits it should be firstly restructured and rewritten. If it is not done every new trait will add a lot of code branches and special cases to actual implementation. This will result in even more unmaintainable application.