Improving Freeciv war AI

Course: Artificial Intelligence For Games

Maciej Hamiga
Mikołaj Kocot

Table of Contents

1. Concept
2. Source code modifications
   1. Modified/new data-structures
   2. Modified/added methods
   3. Main algorithm
3. Testing and conclusion
1. Concept

The purpose of this project, was to improve Freeciv AI mechanisms in the area of war and diplomacy. The idea first presented during class, was to declare wars and conquer depending on geographical borders of a country. Such AI mechanism would surely prove to be an interesting addition to the gameplay - unfortunately after performing some research and source code analysis, we found this idea very difficult to realize.

A new concept was born - after examining Freeciv source code, it became apparent that war AI thinks about war declarations in a completely isolated manner - it takes into account only the attributes of currently analyzed potential opponent (such as distance which we need to travel to attack his country). We decided to force the AI to think globally and keep track of our other diplomatic relations (i.e. declared wars), before making a decision to start a new war. When AI looks for a candidate-enemy, it calculates so-called "war desire". We injected our logic into this process, decreasing war desire for any potential target, when we already are engaged in some conflicts. We also take into account military and economic strength of our enemies, allowing the AI to declare wars to couple of weak targets or 1-2 strong opponents. More detailed description of changes made to the game's source code are described below.

2. Source code modifications

1. Modified/new data-structures

Improving AI players accuracy usually means storing some additional data in player’s connected structures. The same problem had to be handled in this particular situation modifying diplomatic decisions making.

To provide further extension possibilities, as well as keep current project conventions following modifications have been introduced:

Listing 1. Modified ai_data structure, code fragments from ai_data.h header file (changes are marked red)

```c
struct ai_data {
  (...) 
  struct {
    struct ai_dip_intel player_intel[MAX_NUM_PLAYERS + MAX_NUM_BARBARIANS];
    enum winning_strategy strategy;
    int timer; /* pursue our goals with some stubbornness, in turns */
    char love_coeff; /* Reduce love with this % each turn */
    char love_incr; /* Modify love with this fixed amount */
    int req_love_for_peace;
    int req_love_for_alliance;
    struct player *spacerace_leader; /* who is leading the space pack */
    struct player *production_leader;
    int war_counter; /* Keeps track of wars the player is currently engaged in */
    struct ai_war_intel war_intel[MAX_NUM_PLAYERS + MAX_NUM_BARBARIANS]; /* Holds war-related information about our opponents */
  } diplomacy;
};
```

As you can see, the only modifications are related to ai_dip_intel structure nested in most wide AI structure – ai_data.
Two fields has been introduced so far:

**war_counter** – keeps count of wars the player is currently engaged in, in fact count of ai_dip_intel with at_war set to true in war_intel table

**war_intel** – table of ai_war_intel structures, describing war parameters about current player and each of his opponents. Structures are indexed in respect to unique value returned by player_index function.

Next modification worth to mention is the new structure introduced by war_intel table – ai_war_intel. Its code could be seen below. Its main and only responsibility is to store diplomacy state between AI player and his opponent, as well as other war-related parameters, helpful to calculate war_desire attribute.

Listing 2. New ai_war_intel structure

```c
struct ai_war_intel {
    signed char at_war; /* Are we at war with this player */
    int player_power; /* How powerful we consider this player */
};
```

Currently implemented fields (the list can be extended in the future to provide even more accurate war_desire calculation):

**signed char at_war** – keeps information about current war state with referenced player

**int player_power** – holds relative power determinant of referenced player based on his military abilities or -1 value if player is not alive any more. The bigger the player_power attribute is, the lower is the desire to declare a war against him.

2. Modified/added methods

To keep things simple, some additional methods have been provided for basic operations with new structures. All of them could be found in ai_data.h header file, with implementation in ai_data.c. Let’s see some declarations:

```c
struct ai_war_intel* ai_war_get(struct player* pplayer, struct player* aplayer) - the simplest one, with only duty – return ai_war_intel pointer from pplayer’s ai_data structure, connected with aplayer. Both player must be of player pointer type; Encapsulates all logic necessary to retrieve proper row from war_impel table.

void ai_set_player_war_state(struct player* pplayer, struct player* aplayer, signed char at_war) – sets war state of pplayer and aplayer in pplayer’s ai_data structure. at_war allowed values are TRUE and FALSE, in future some additional war-states addition is possible. Notice: this method modifies one structure at the time.

void ai_set_war_state(struct player* pplayer, struct player* aplayer, signed char at_war) – similar to ai_set_player_war_state method, although modifies both players ai_data structure instead.

void ai_dispose_ai_war_intel(struct player* pplayer, struct player* to_be_disposed) - sets every attribute of to_be_disposed’s ai_war_impel structure in pplayer’s ai_data to neutral values, adjusting war_counter to proper value, at the same time. Usually used after to_be_disposed player has been killed.
```
Apart from hooking up into players lifecycle in various points of Freeciv code, we also modified the function that is responsible for calculating war desire for AI opponents. The function can be found in advdiplomacy.c file, and has the signature presented below:

```c
static int ai_war_desire(struct player *pplayer, struct player *target, struct ai_data *ai) - where pplayer is current player, target represents the opponent we currently take into account, and ai structure stores current AI information. Functions returns calculated war_desire value against target.
```

3. Main algorithm

Listing 3. Algorithm responsible for decreasing war desire, located in `ai_war_desire` function.

(...)

DIPLO_LOG(LOG_VERBOSE, pplayer, target, "warlog: War want before war counter adjustment: %d", want);

    ai->diplomacy.war_intel[player_index(target)].player_power = player_power;
    /* Update player power with calculated value */

    /* Take into account number of wars we are currently involved in, and the power of our opponents */
    if (ai->diplomacy.war_counter > 0) {
        int abs_want = abs(want);

        double normalized_war_counter = (double) ai->diplomacy.war_counter / (double) MAX_NUM_PLAYERS; /* Normalize counter to 0.0-1.0 range */

        int war_intel_modifier = (int) abs_want*normalized_war_counter;

        /* Let us count overall power of players that we are fighting with */
        int overall_power = 0, opponents_count = 0;
        players_iterate(aplayer) {
            if (ai->diplomacy.war_intel[player_index(aplayer)].at_war) {
                opponents_count += 1;
                overall_power +=
                    ai->diplomacy.war_intel[player_index(aplayer)].player_power;
            }
        } players_iterate_end;

        if (opponents_count > 0) {
            war_intel_modifier += overall_power / opponents_count; /* Average power of opponent */
        }

        want -= war_intel_modifier;
        /* Subtract calculated modifier from overall war want */
    }

    DIPLO_LOG(LOG_VERBOSE, pplayer, target, "warlog: War want after war counter adjustment: %d", want);
(...)

**player_power** variable is calculated in function's part executed before the fragment presented on listing 3. It takes into account enemy wealth, units, city defenses.

Our algorithm normalizes **war_counter**, so that it represents percentage of players that we are fighting with. The overall war want computed by the function is then decreased by this percentage.

Additionally, we iterate over every player in game, counting number of opponents and summing up their power. In the next step, we once again decrease war want, by the average power of our current enemies.

### 3. Testing and conclusion

We performed some basic testing, using Freeciv server autogame feature, and also by playing some sample games.

Autogame test were performed on AI hard level of difficulty, with 7 players and specific game and map seeds, to ensure no randomness. We run several test games, with our modification compiled into the server, and without it.

We observed a **15-20% less aggressive AI gameplay**, in terms of overall, absolute war incidents. But what is even more interesting, we looked at the way each separate AI plays, and observed a significant improvement: **With our modification, most AI players where engaged in mostly 1-2 conflicts at a time, with rare peak value of simultaneous conflict at level of 3.** Standard AIs were much more aggressive, with most of them engaged in 3 or even 4 conflicts at one time, with peak value of even 6 simultaneous conflicts - this means that a standard AI player was fighting with all other players on the map!

During real gameplay, the difference was not obvious, but one could feel that AI is less aggressive and more focused on development. However, it still acted offensively when the right time has come.

Concluding, we can sum up our attempt to improve war AI as successful - computer players behave in a more realistic way, which we think should be a priority in civilization-like game. We can only add, that a recent release of Freeciv, dated 7th December, focused mostly on "making AI a little less aggressive" what we think proves our point :)