



Contents

1	Strategy	2
1.1	Diplomacy	2
1.1.1	Introduction	2
1.1.2	Relationship score	3
1.1.3	Example scenario	3
1.1.4	Functions definitions and guidelines on balancing	4
1.1.5	Characteristics of the model	5



Chapter 1

Strategy

Strategy INFO TODOj

1.1 Diplomacy

1.1.1 Introduction

Diplomacy is currently done by StrategyManager along with other Conditions. There are three special conditions created for this purpose:

- ConditionAllied
- ConditionNeutral
- ConditionHostile

What they do is check diplomacy against each of the players. Of course it is guaranteed that any two players will be either allies, neutrals or hostiles. Conditions then creates a “hostile_player”, “neutral_player” or “allied_player” action that is then passed to BehaviorManager. BehaviorManager asks its BehaviorComponents whether any would be interested in “solving” given case. Three special BehaviorComponents are created for this purpose. Each of them is characterized by different traits, so naturally they value different things in other players in game.

1. BehaviorEvil

- (a) Things it likes in other players: **strength, large terrain, wealth.**

2. BehaviorNeutral

- (a) Things it likes in other players: **strength.**
- (b) Neutral about **wealth, terrain size**

3. BehaviorGood

- (a) Things it likes in other players: **weakness, small terrain.**
- (b) Neutral about **wealth**

Values taken into account are:

1. strength - $((CHP(ai))/(CHP(player)) * ((lenShips(AI))/(lenShips(player))))$, where $CHP(X)$ is cumulative fighting ship HP of given player X, while $lenShips(X)$ if amount of fighting ships by given player X.
 2. terrain size - $(TSize(ai)/TSize(player)) * (lenIsl(ai)/lenIsl(player))$, where $TSize(X)$ is amount of ground tiles owned by given player, and $lenIsl(X)$ is amount of separate islands owner by given player.
 3. wealth - $(Wres * Res(ai) + Wgold * Gold(ai))/(Wres * Res(player) + Wgold * Gold(player))$ - Where $Res(X)$ is total worth of resources by given player, and $Gold(X)$ is amount of Gold by given player. $Wres$ and $Wgold$ are weights each values has in equation. Currently $Wres = 0.75$ while $Wgold = 0.25$, so resources are valued three times as much as raw gold.
-



1.1.2 Relationship score

Note that every value **strength**, **terrain size** and **wealth** is not a concrete number, but a balance between AI and other player. Theoretically, values of balance are in range of $(0, \infty)$, where values like 0.5 mean that other player is twice as strong/big/wealthy as AI. Similarly balance value of 3.0 would mean that AI is three times as strong/big/wealthy as the player it was matched against. To make it simpler, this value is first trimmed to range of 0.1 and 10.0 (to avoid handling zeroes and infinities we assume *balance* > 10.0 is 10.0 and *balance* < 0.1 is 0.1). Later, this trimmed value is translated into continuous scale of -10.0 to 10.0, where -10.0 states that given player was close to 10 times weaker than AI (values near 10.0 stating the exact opposite).

So what we have so far are three values from a range of $(-10.0, 10.0)$ stating how given player compares to AI. As said previously, AI can have any of the three Behaviors defined, that handle diplomacy (**Evil**, **Neutral** and **Good**). To simulate “preference” of each behavior, they have weights defined for each of the balance values. Evil behavior can have these settings to simulate fondness for strength, large terrains and wealth:

- ‘power’: -0.6
- ‘terrain’: -0.3
- ‘wealth’: -0.1

Negative weights mean that in case where enemy is stronger than AI (negative balance), AI takes that as an advantage (higher chance to form an alliance with player) Final value is calculated by multiplying weights with corresponding balance values. All weights sum up to values between -1.0 and 1.0 to make sure the final value (*weights * balances*) is between -10.0 and 10.0.

What we have right now, is a single real value describing how given Behavior (**Evil**, **Neutral** or **Good**) respects, or “likes” given player. What we would like right now, is to simulate a scenario where low values (< 0.0) tend to start wars with given AI, while high values (> 0.0) make AI form an alliance with given player. We could two single threshold values (e.g. *war* = -5.0, *peace* = 6.5) and then have some randomness that decides whether to start a war yet or not. In this case value of -10.0 and -5.0001 would make no difference in behavior, so it’s not the best solution. Instead we do something else: Each of the three diplomatic Behavior has certain parabolic functions attached to it for each of the types of diplomacy settings against player. In other words: BehaviorEvil (and the remaining two) has three functions defined for cases where enemy is **allied**, **neutral** and **hostile**. That gives us 9 different functions to define totally. Graphical example of that can be see on image 1.1. Notice that there’s no function defined for “NEUTRAL” action. More on that later.

1.1.3 Example scenario

I know I didn’t give out details yet on how the functions work, but let’s start out with an example. Let’s say *relationshipscore* was already calculated against given player (as described earlier) and is equal to **4.36**. Let’s also assume it’s the case where other player was **hostile**. See at image 1.2 to have a grasp of how functions could be defined. Notice that player is hostile at that moment, so it should be easy to switch back to peace. We don’t have hostility function defined here, because it’s the current state of diplomacy between players. “Doing nothing” is also an option when handling diplomacy, so turning hostile again wouldn’t really differ from that. Settings show at the figure fit more to Good-natured behavior since it’s fairly easy to stop war (all you need is *relationshipscore* slightly larger than 0.0).

When handling diplomacy, four possible actions can be done:

- WAR - declare war with given player
- PEACE - form alliance with given player
- NEUTRAL - declare neutrality towards given player
- WAIT - do nothing

Additionally, two random variables may be generated:

- First one states whether any change will be done (see fig. 1.3) or not. It’s equivalent to determining whether random point on a vertical line positioned at *relationship_value* would hit yellow or non-yellow pixel.

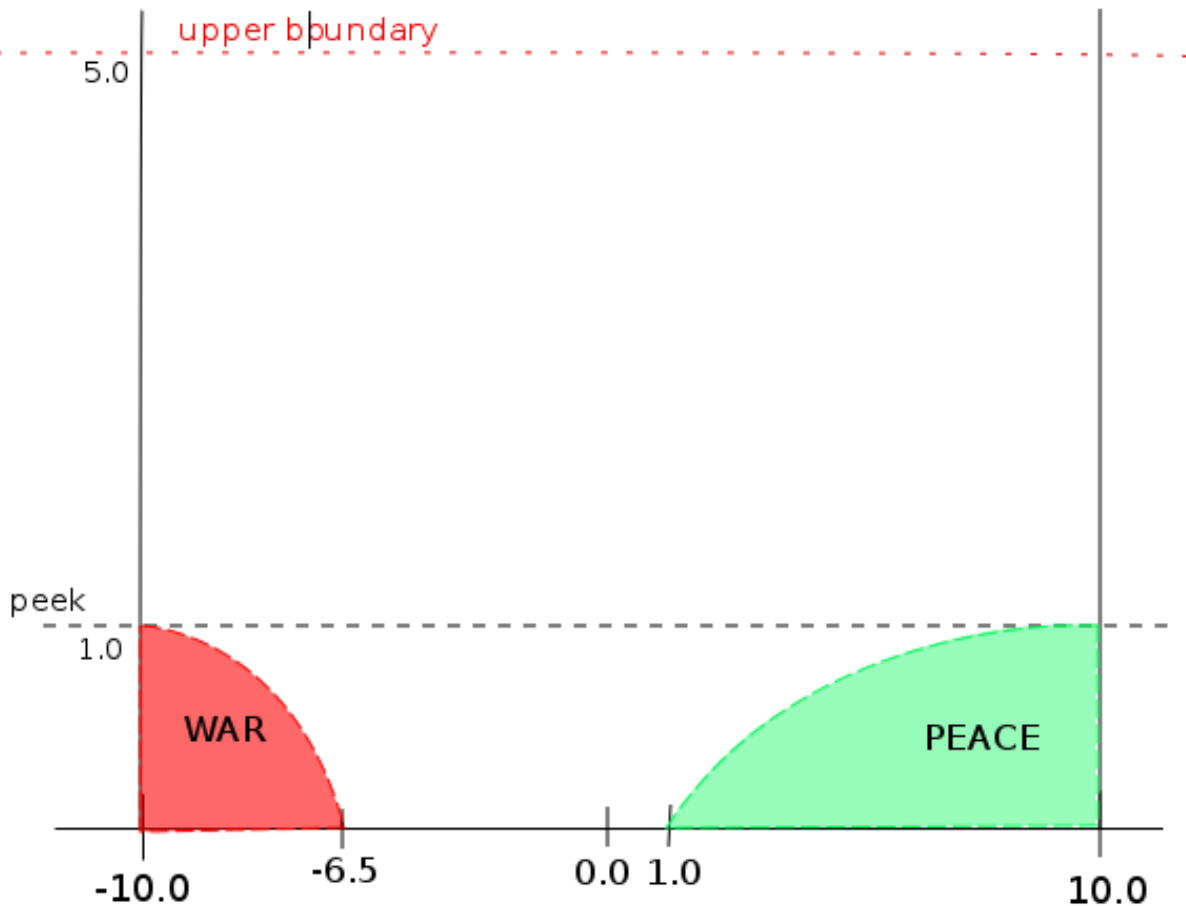


Figure 1.1: Diplomacy settings for “neutral_player” action.

- If an action was selected (as opposed to waiting), second random variable states which type of action is selected. That’s done only when the regions intersect (as shown on fig. 1.3). Two red spots show probabilities of **PEACE** and **NEUTRAL** actions. With *relationship_value* fixed at 4.36, **NEUTRAL** has higher probability to take place.

1.1.4 Functions definitions and guidelines on balancing

As explained before, we have to define 9 sets of parameters totally. This is done for each of the functions (**hostile_player**, **neutral_player** and **allied_player**) for each of the components (**BehaviorEvil**, **BehaviorNeutral** and **BehaviorGood**), as static parameters in class definition. Parameters given for each functions are:

- **mid** - “Midpoint” or position of the “peek” on X axis (this is fixed for hostile (WAR) and ally (PEACE) functions on -10.0 and 10.0 respectively).
- **root** - point on X axis that’s the intersection point of OX axis and function itself. This is somewhat equivalent to *threshold* value.
- **peek** - how tall should the function be, i.e. what’s the maximum value it can achieve at peek point. By default this is 1.0.

Additionally there’s also *upper_boundary* value which determines how big the probability to *WAIT* is. Function parameters are provided in each of the BehaviorComponents (Evil, Neutral and Good) in form of a python dictionary, e.g.

```
1 parameters_hostile = {
2     'neutral': {'mid': 0.0, 'root': 2.0, 'peek': 0.3, }
```

```

3         'ally': {'root': 4.0 }
4     }

```

To give you a feeling how each parameter alters the chance for given action to be executed, let's use blue function *NEUTRAL* in 1.2 as an example. High “peek” would make the function taller, increasing its chance to be called when the parameter “relationship_score” is somewhere between 0.4 and 6.2). The roots (0.4 and 6.2) would remain the same. If you were to alter the “root” parameter, you'd make function broader. Function shown on the figure has parameters 'mid' and 'root' equal to 2.9 and 0.4 respectively (notice that parameters like 2.9 and 6.2 would return the same function). Notice that 'root' parameter is simply a point on OX that the function should intersect, so it's directly connected with it's 'mid' value.

1.1.5 Characteristics of the model

Model comes with certain advantages and disadvantages.

- Advantages:
 1. Relatively few values are needed to define by hand in order to have good results.
 2. upper_boundary value which allows to “globally” make all actions appear less or more often
- Disadvantages:
 1. As it comes with heuristic models, it's not perfect.
 2. Quadratic function (x^2) is used, currently it's not possible to provide other functions, but it's easy to add.

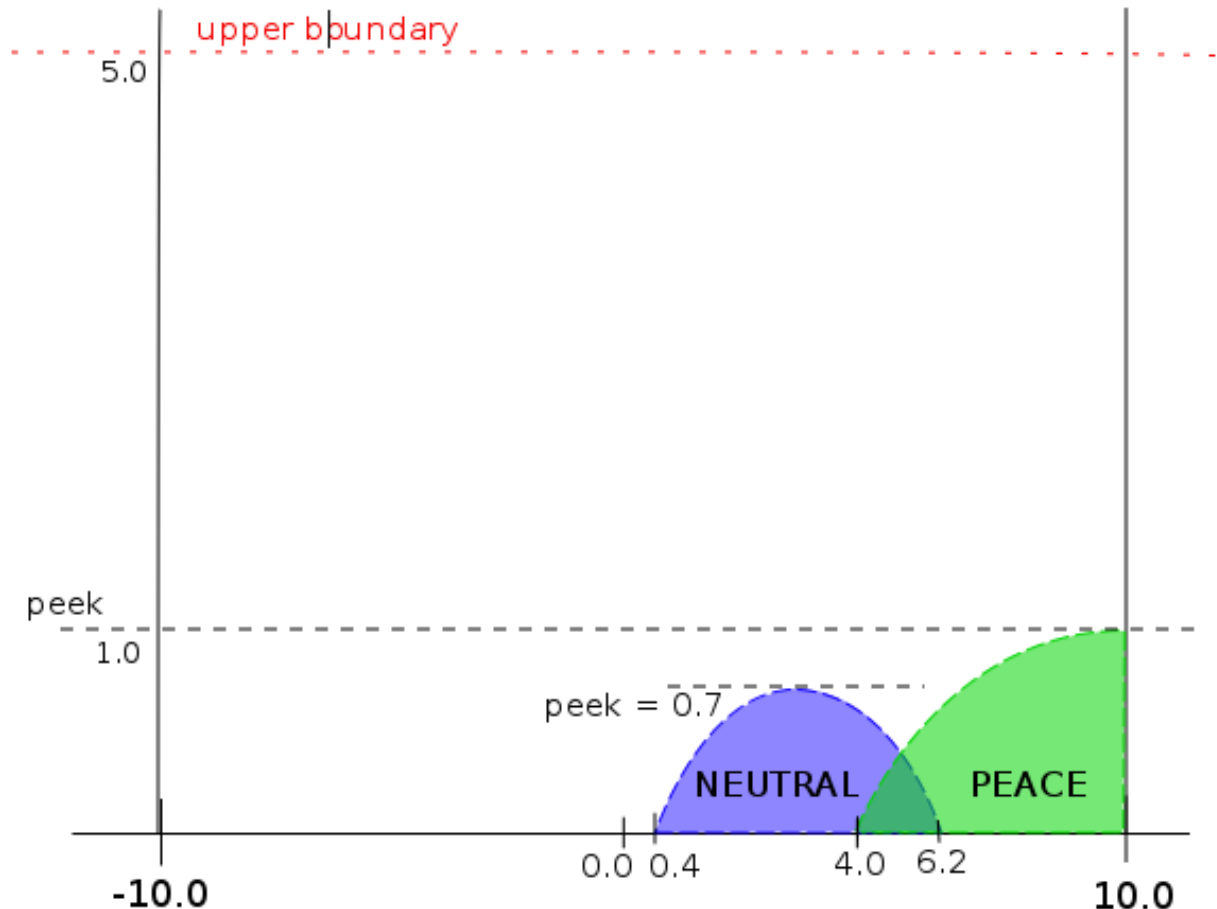


Figure 1.2: Diplomacy for “hostile_player” action.

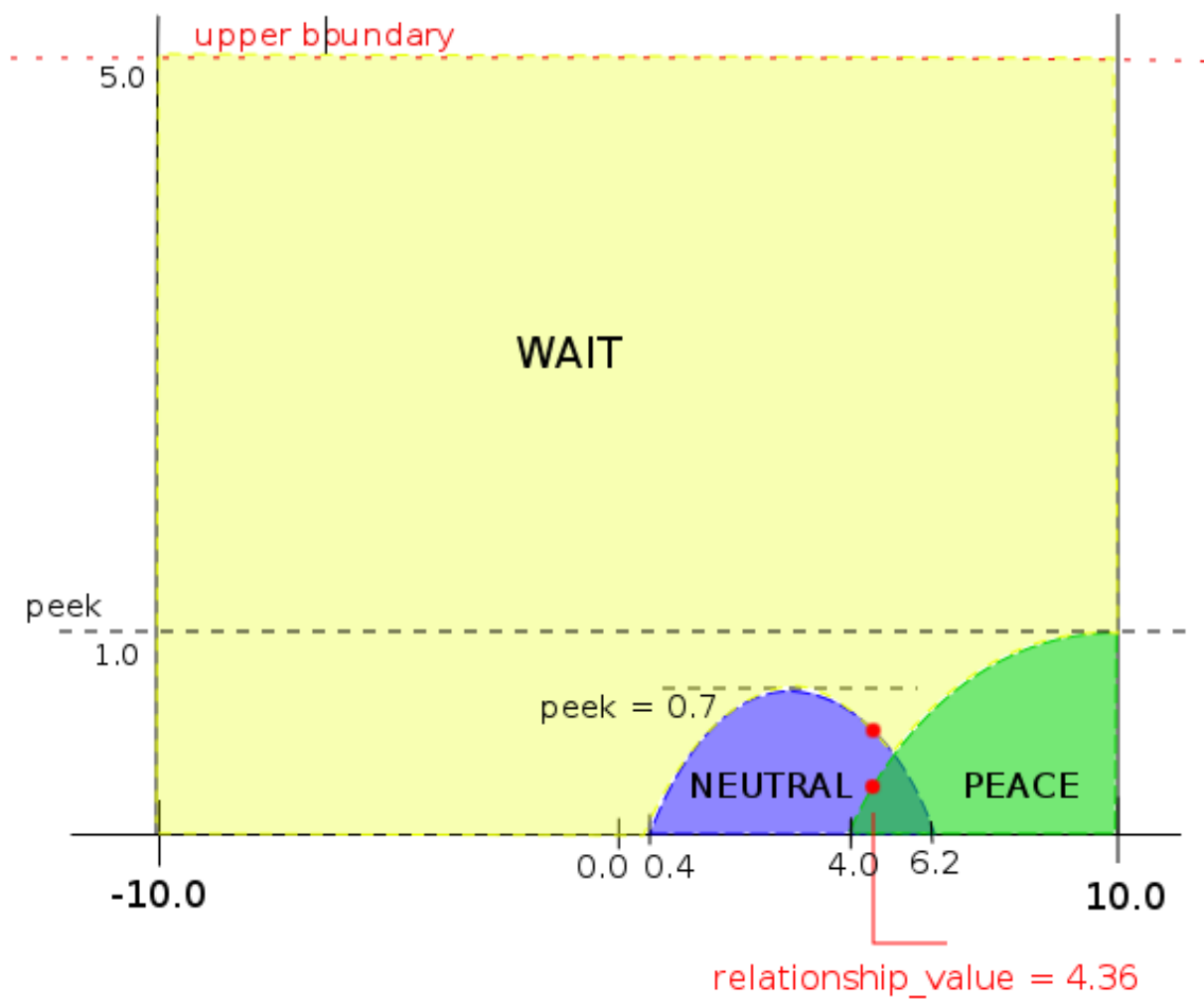


Figure 1.3: Diplomacy for "hostile_player" action.