Some modifications that would probably make life simpler for our learner.

Reference: Samuel, A. L., 1959. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development 3 (3), 210-229.
URL http://www.research.ibm.com/journal/rd/033/ibmrd0303B.pdf

I was a little puzzled by the behavior of our program the other day. I mean: it is really cool, but it definitely plays in a rather… convoluted way. One gets the impression that the program can recognise when the game ends, but does not know how to give a direction to its playing. Of course, 20 games are not enough to expect the player to improve very significantly, yet one may ask himself: What if the program did not improve even after – say – 1000 games? Should we keep on training it, or should we suspect that perhaps it just is not learning what it should?

It probably would have been better to read this paper at least twice before starting designing the program. But of course no one ever has the time to do everything they would ideally do. Given that you guys are certainly better suited to coding, I thought it was my duty to invest some time in giving this doubt some more consideration. I went through the paper referenced above again, and there are a few points I'd like to draw your attention to. I think they can make a dramatic difference in the learning speed of the program (or perhaps even make the difference between learning and *not* learning).

Definition of "Goal" and "Dominant criterion"

The program needs to be informed about two things:

(a) The final goal of the game:

(b) What it means *in general* to put itself in an advantageous position with respect to the opposing player (the *dominant criterion*).

In our case, the goal is to join the two opposite sides of the board the player owns. Since this can happen only once in the game, it is not a feature and is tested separately. We did this correctly. In checkers, the dominant criterion is the *relative piece advantage*: a move gets a positive score if it leads to reducing the number of pieces of the opponent (and a negative score in the opposite case). What is the equivalent criterion in twixt? My proposal would be: the difference between my f_4 and my opponent's f_4. The fact that we did not design this feature may make it harder for our learner to actually learn the game.

Immutability of the weight of the dominant-criterion feature.

In our reference paper, the dominant-criterion term of the weighted sum has a special treatment. Unlike all the other terms, its weight is fixed: The program is not allowed to modify it during the game. The rationale underneath this strategy is that this feature *defines the task itself* of the player: It is not acceptable for the player to decide to change its task. The task is defined by the game, not by a calculation of the player given a certain state of the board. This is probably a good point and we should take it into account.

Training strategy

We have already discussed this a bit. I know your ideas are slightly different from this, but I think it is good to summarize what Samuel found was an effective strategy for training its player (besides, of course, playing against a good human player).

When the program plays against itself, the two electronic players (Alfa and Beta) are not exactly the same. Alfa updates its evaluation function every time it makes a move, while Beta uses the same evaluation function for the duration of any one game.

At the end of each game the ability of Alfa relative to Beta is evaluated. If Alfa won, or is judged to be ahead (in case of a draw), Alfa's scoring system is given to Beta. If Alfa loses or ends a draw behind, it receives a "black mark". If Alfa receives three black marks (it is not clear whether Samuel refers to three in a row, or he means that at any given time Beta is three wins ahead), a drastic change is made: the coefficient of the leading term in the summation is set to zero. The rationale is that, evidently, Alfa is on the wrong track and that term has the greatest responsibility for this.

Limitations of the perceptron

If the player still does not seem to be able to improve its game after adopting these "tricks", we may have to consider that perhaps the perceptron is too simplistic a model for the complexity of our task. I am not an expert, but have noticed that usually the perceptron (with its learning rule as we implemented it) is presented as a device that can perform rather simple ("linearly separable") classification tasks. It could be that we need to move on to something more complex like a 2-layer neural network, if things start looking really bad.