Thomas Lum
4/23/2015
CMPU 2015
McCarthy

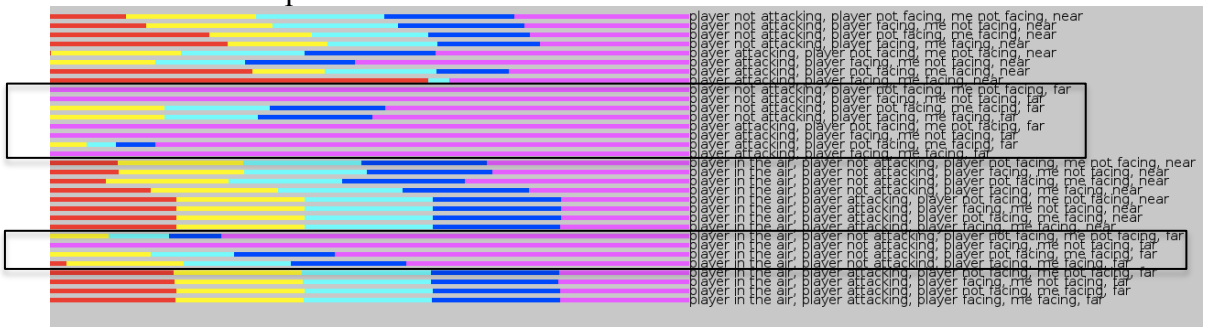
Simulated Annealing as Applied to Semi-Continuous Strategy Building AI

The big picture goal of this project was to implement a system of AI decision making that would adapt to perform ideal actions according to the very loosely defined function that was the play-style of the player, similar to finding the optimum value through simulated annealing on a function.

The game consists of the player controlled character and 10 other enemy rectangles that all draw from and add to the same collective brain for individual decision making, hence the title Swarm. All actors can move left, right, jump, and attack, though movement for the AI is measured in whether it is moving towards or away from the player rather than left or right. The goal for both sides is to attack and not be attacked and the score in the center of the screen is the number of frames the player has attacked minus the number of frames the player has been attacked.

There are five different parameters, or perceptible actions that the AI are aware of: whether or not the player is within the range of attack (near/far), whether the AI is facing the player, whether the player is facing the AI, whether the player is in the air, and whether the player is attacking or not. As a result there are 32 different decision weight graphs, one for each permutation. Each graph represents a relative context (RC (often refered to in the code as the wn or weight number)) a specific AI can be in at any time, or a particular confluence of relative perceptible actions. Each decision weight graph consists of weightings of the chance to perform one of five different performable actions the AI can take: attack, walk towards, walk away, jump, and stay still, which are color coated red, yellow, cyan, blue, and magenta respectively. As time passes, the chance that an AI will use the weighting graph as opposed to simply deciding equally randomly what to do will decrease. While each individual AI can be in a particular RC, the graphs and the weightings are shared amongst all the AI.

There are five factors that affect the weighting of the graphs. The first is tiredness or boredom: any RC the AI is currently in will slowly have the weights for all actions besides being non-still decrease. This is based on the idea that if the AI cannot perform any action that is beneficial, it should simply not exert any excess “energy”. Another way of imagining this is that the AI becomes “bored” of taking actions that offer no benefit. For example:



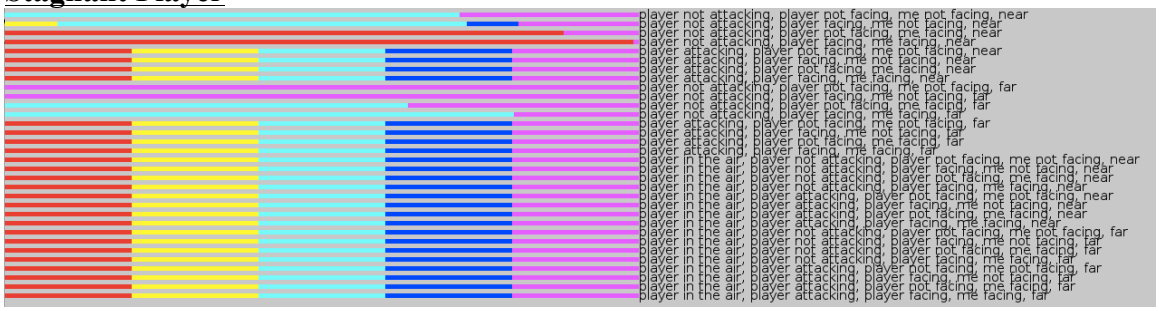
In the above RC list[1], actions where the player was far away from the AI (as marked by the black rectangles) had much higher weightings for staying still, as they were in no immediate threat nor in any position to benefit themselves. All other weight shiftings are based on reward and punishment functions that accordingly increase or decrease the weight of the current action the AI is using in the current RC it is in as well as the previous action taken by the AI (more on this later). In a similar vein to the tiredness factor, if an AI attacks but does not hit its target, it is punished. For instance, if the AI is facing the wrong direction and attacks, or is far away from the opponent and attacks, the weight of performing that action is decreased, or in any other scenario where the player is not hit. Being hit by the player is also punished, and the according action performed while being hit is lowered in weight. The two actions that can cause a reward and an increase in weighting are when the AI successfully lands an attack and when the player attacks but does not his the AI.

However, it was not simply enough to keep track of what the current action being performed during the time of punishment was, as actions are often chained together meaningfully rather than performed in isolation, such as moving toward the player then attacking. Thus, along with this, the previous action and the previous RC are recorded, and given the a slightly less significant alteration in weight according to the punishment/reward of the current action.

Actions are performed after a random number of frames between 20 and 50 or when the RC changes. This segmentation of actions into chunks arguably reduces its continuous nature, however player actions remain continuous, and so general heuristics need be "created" by the AI that fit a dynamic range of situations.

In order to test that the AI was adapting to different play-styles, three main methods of playing were analyzed. The first was to simply stay in the middle of the screen and do nothing. The second was to jump around frantically (this mode can be seen still by pressing the j key). The final case was to simply play as I would normally, attacking and jumping and moving as best I saw fit.

Stagnant Player

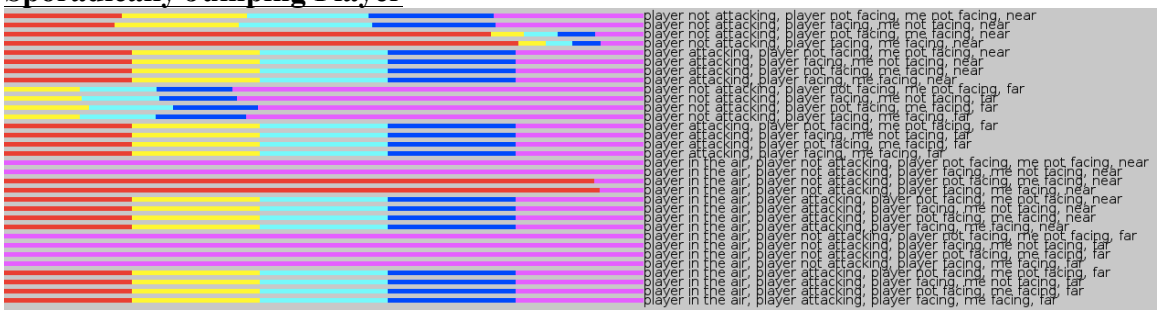


As expected the stagnant player produced a very clear dominant strategy for the AI. In this case, the bars where the player is in the air (the bottom half) as well as where the player is not attacking should be ignored, as those RCs are never entered. Because the player offered no threat, the primary actions were attacking and staying still or chasing

[1] One consistent trend that is found amongst many of the graphs is a lack of weight changes on particularly rare RCs, such as any context where the player is both in the air and attacking, as attacking takes time to execute, and is thus not very practical. As such, from here on out, the bottom four bars will be omitted unless they present outstanding occurrences.

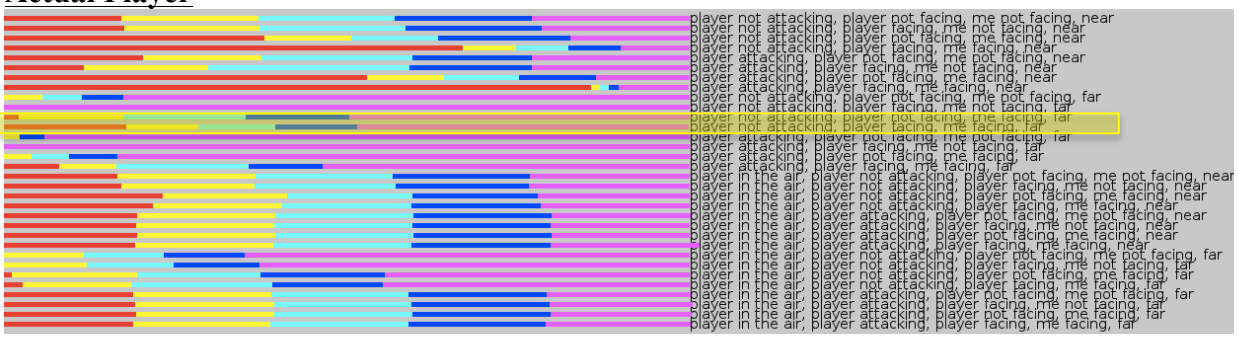
the player if they were far away, as well as approaching the player if the AI is facing the wrong direction.

Sporadically Jumping Player



As was the case with the above scenario, when the player was frantically jumping, all attack related RCs were essentially ignored. This produced very similar results to the previous play-style, with primarily RCs with the player being airborn properly weighted, as the player was only ever on the ground for a split second. In this scenario, because the players movement was sporadic and literally random, all scenarios where the AI was far from the player skewed towards or became entirely the decision to stand still, as moving towards or away would not guarantee leading to a reward any more than standing still. That said, the AI still heavily favored attacking when facing the opponent rather than away. Ideally this case would have produced a tendency for the AI to jump and attack in sequence, however the intervals where such a method is effective was simply too small to produce noteworthy weightings, as jumping and falling is fast and attacking is not immediate. Perhaps with better memory and altered physics this result could be produced.

Actual Player



Perhaps most intriguing was this weighting of the AI that was created when I was playing. As in previous cases, attacking when near and facing was preferred over other cases, and scenarios where the player was far often led to staying still. I did not tend to jump, so the second half of the data is less significant. The AI seemed to favor attacking when I was facing the AI, perhaps because I would often stand before an AI and attack while being attacked myself for periods of time. There is also a high tendency for moving towards the player in the RC on row 5, which is where the player is near but facing the wrong direction and the player is facing and attacking.

However the most interesting weighting to occur is the one highlighted in the yellow bar above and enlarged below.

The context for these bars is when the AI is facing the player, and the player is on the ground, not attacking, and far away. However, the difference between the top and bottom bar, is that the bottom bar is for the RC where the player is facing the AI, and the top is for when the player is facing away. Though one would expect that the two would be the same or comparable, the second red attack bar is higher because of the reward that is gained from the pattern acknowledge by the AI: even if the player is out of range, when the player is facing the AI, it has been shown that the player will continue moving forward to attack in that scenario. Thus if the AI attacks now, the player will essentially run into the attack. Through simple trials and only the perceptible actions, and rewards and punishments defined above and the weighting algorithms in place, the AI had learned to *pre-emptively attack*.

Future extensions and examinations of this methodology would benefit from increased analysis and experimenation of memory systems and flow of actions, as well as more perceptible actions and performable actions. As well, proper rebalancing of weights and a larger sample of play-styles would increase the strength of this analysis.