1 Two-system Q Learning

A common feature of games is menus, where players must chose game options, customizations, etc. Usually, the game's menus are not like the game itself, and consequently many learning agents have trouble "playing" the menus. To rememdy the issue, we will introduce a second learning agent which learns specifically to understand and "play" the menus.

To help the system along a bit, we will automatically handle switching between the game agent and the menu agent. The game's menus and the game itself are composed of different *screens*, each of which we will assign a number. To illustrate the concept, consider a toy game example whose title screen is associated with the number 0, the options screen is associated with 1, and the actual game screen is associated with 2. The policy π of the system will be composed of two subpolicies: π_m for menus and π_g for the game. Let n_s be the number of the currently active screen and x be the currently observed state, then

$$\pi(x) = \begin{cases} \pi_m(x) & n_s = 0 \text{ or } n_s = 1 \\ \pi_g(x) & n_s = 2 \end{cases} \tag{1}$$

Since each policy deals with different kinds of state and action (the menu policy must understand how to navigate whereas the game policy is fairly simple for our choice of game), we can minimize confusion in each policy by completely separating their state and action spaces. The menu policy needs only information like the screen number, the currently selected menu option, the number of upgrade points available, and it can only take actions like moving the selection up/down/left/right, clicking, and returning to the previous screen. The game policy on the other hand has actions like changing the direction of the launcher and increasing the force of the launcher, and the observations it can make only concern the current direction and force values. Using the same naming scheme as for policies, we call the state and action space pairs of each policy (S_m, A_m) and (S_g, A_g) respectively. The menu policy is $\pi_m : S_m \to A_m$ and the game policy is $\pi_g : S_g \to A_g$.

2 Tractable Training and Live Playback

Our system is scored by preferences on segments; we have a magic function $\mathcal{R}(\sigma)$ which forms preferences by asserting $\sigma_i \succ \sigma_j$ if and only if $\mathcal{R}(\sigma_i) > \mathcal{R}(\sigma_j)$. Right away there's a problem: we have no way to decide when a segment ends. The menu policy could select menu items until the end of time without stopping, and that would form a valid segment. We can't tractibly let the policies keep going ad infinitum if we wish to compare their performance.