Local Behavior Selection

Gal A. Kaminka

Local Behavior Selection

Local/distributed evaluation

SELECT():

- Each behavior has associated *activation function* $active(b)$
  - Captures "how much" the behavior should run
- Selection mechanism is very simple: $argmax\, active(b)$
- Once selected, behavior is executed immediately

TERMINATE() (two options):

- Terminates when loses competition
  - $active(b)$ continues to be evaluated
- Terminates when releases control
  - $active(b)$ is reset when b is ready to be finished

Types of activation functions

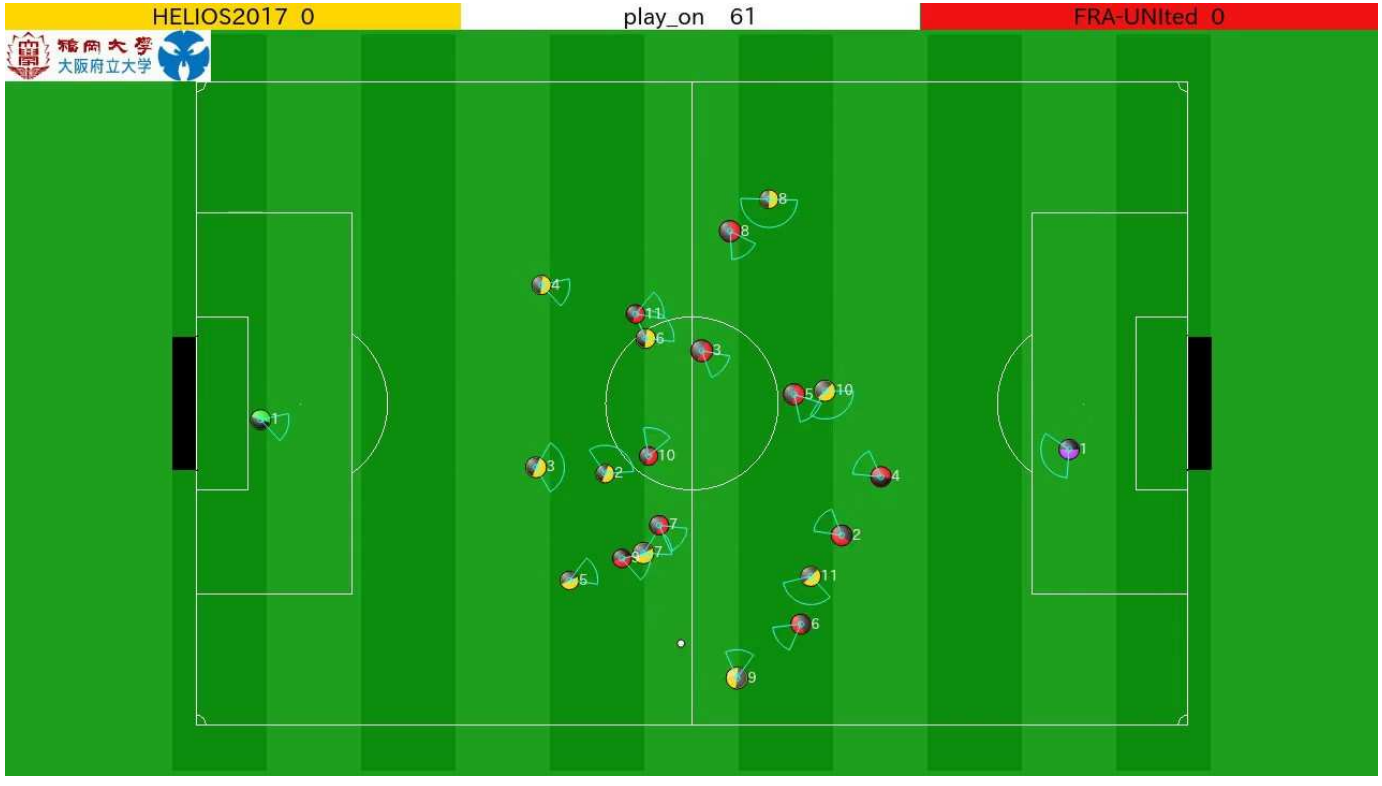
Many different factors possible:

- Usefulness to agent (**value**, **utility**)
- Urgency (**priority**)
- Likelihood of success (**success probability**)
- Matching current state (**applicability**)

Can of course combine these (e.g., **utility** $\times$ **success probability**)

Case Study: ChaMeleons 2001 RoboCup team

- Carnegie Mellon team in RoboCup 2D simulated soccer competition
- Each team: 11 separate programs (coach agent optional)



The ChaMeleons 2001 HandleBall() Arbitrator

Priority	Descriptor	Success Probability Threshold
1	*shoot_on_goal*	.8
2	*coach_pass_for_shot*	.7
3	*pass_for_shot*	.8
4	*pass_forward*	.75
5	*dribble_to_goal*	.75
6	*dribble_to_corner*	.8
7	*pass_to_less_congested*	.7
8	*coach_pass_forward*	.8
9	*pass_to_closer_to_goal*	.75
10	*pass_to_better_path_to_goal*	.8
11	*shoot_on_goal*	.6

Table 1. One Possible Priority Level Ordering

Select behavior *b* such that:

- priority class is maximal (minimal value), **and**
- *b*'s probability of success is over priority class threshold, **and**
- *b*'s probability of success is maximal within the priority class

What's good about Activation-Based selection

- Soft goal or subgoal achievement
- In some cases, easy to give a number
- Decision-mechanism itself is fast and simple
- Activation, even of non-selected behaviors, gives useful information
  - e.g., ranking behaviors for selection
  - e.g., upcoming potential behaviors
  - See Behavior-Networks in the T.A. class

Problems in Local Evaluation (Activation Functions)

- Thrashing is a common issue
  - small changes in activation, near *argmax* selection thresholds
  - especially with high-frequency re-evaluation

Problems in Local Evaluation (Activation Functions)

- Thrashing is a common issue
  - small changes in activation, near *argmax* selection thresholds
  - especially with high-frequency re-evaluation
- Does not scale well in terms of software engineering:
  - No context to managing selection
  - No memory, no “special cases”

Problems in Local Evaluation (Activation Functions)

- Thrashing is a common issue
  - small changes in activation, near *argmax* selection thresholds
  - especially with high-frequency re-evaluation
- Does not scale well in terms of software engineering:
  - No context to managing selection
  - No memory, no "special cases"
- Lots of Computation, Memory
  - Each behavior needs to do own groundings
  - Each behavior needs to do its own evaluation
  - Often with high-frequency

Looks simple, but really isn't

- Always looks easy and elegant in design
- In my experience, invariably leads to *number hacking*
  - extending activation range, tweaking
- CPU hungry
- No state, only "selfish" view