Aalto-yliopisto
Aalto-universitetet
Aalto University

ELEC-E7852

Computational Interaction and Design

Assignment A5c

Computational Rationality

Aitor Urruticoechea Puig
aitor.urruticoecheapuig@aalto.fi
Student N°101444219
December 2024

Acknowledgement

Parts of the code implemented in this work were generated with the assistance of GitHub Copilot. None of its proposals were, however, employed without refinement and necessary tweaks to adapt it to the nuances of the tasks at hand; making it impossible to identify and subsequently mark which lines of code were human or machine-made, for many were the fruit of the combination of both.

Contents

List of Figures

1 Introduction and Reward change rationality

Now for the third edition of Assignment 5, the goal is to play around with the concepts of reward engineering with the objective of creating an agent that is more "human-like" when taking the go-no-go decision. Right, now, there are quite a few factors that make the agents (both noisy and not) quite unrealistic. For this assignment, we will focus on:

- **Increased Penalty for Collisions**: Humans tend to avoid collisions at all costs. Collisions should have a significantly higher negative reward.
- **Moderate Penalty for Excessive Waiting**: Overly cautious behaviour (i.e., waiting indefinitely) is not realistic either, so add a slight penalty for inaction beyond a reasonable timeframe.
- **Safety Margin**: Introduce a positive reward for maintaining a safe distance when crossing. In that sense, one can penalize decisions based on how close the vehicle was at the time of action. For instance, riskier actions (i.e., crossing when the vehicle is very close) result in higher penalties, even if there is no collision.

2 Implementation

The increased penalties for collisions and excessive waiting have semi-direct implementation in the existing agent class, as it already has those provisions. Thus, in the original Jupyter notebook, one only needs to specify that:

```
# Create a new instance of the environment with modified penalties
from driver_agent_physics import driver_agent_physics

# Initialize the updated agent with new penalties
updated_agent = driver_agent_physics(
    physics_env=physics_env.physics_env(),
    goal_reward=2,  # Slightly reduced reward for task completion
    collision_reward=-100,  # Stronger penalty for collisions
    observation_var=0  # Full observer
)
```

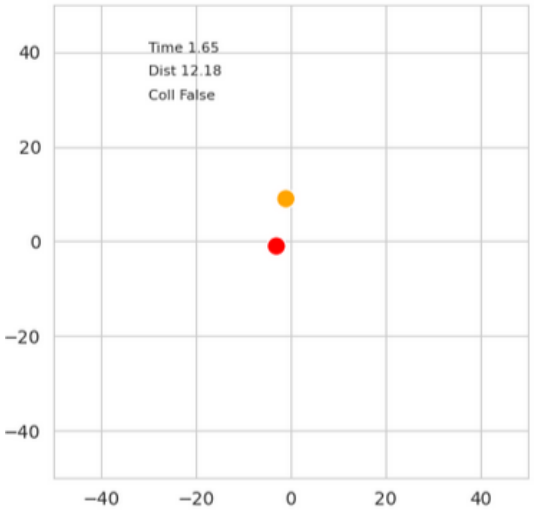
Regarding the safety margin, its implementation is not complex, but needs to be done in the agent class, as there are no provisions for specifying it from its possible inputs.

```
# Code added in the agent Class, step function
if action == 0: #NO-GO
    self.env.tick()
    self.ticks += 1
    # break if nothing ever happens
    if self.ticks > self.max_ticks:
        #print("Too many ticks")
        self.reward = self.collision_reward
        self.done = True
        trunc = True
    if self.env.get_distance() > self.max_distance:
        self.reward = self.collision_reward
        self.done = True
    # Bonus for waiting in high-risk situations
    if not self.done:
        current_distance = self.env.get_distance()
        if current_distance < 3:  # High-risk scenario
            self.reward += 0.5  # Encourage waiting
if action == 1: #GO
    # Did we wait for the other car before going?
    if self.env.veh2_turn_pos[1] < self.env.veh1_straight_pos[1]:
        self.waited_before_go = True
    self.distance_at_go = self.env.get_distance()
    self.done = True
    self.collision, _ = self.env.simulate_go()
    if self.collision:
        self.reward = self.collision_reward
    else:
        self.reward = self.goal_reward - self.penalty_per_tick * self.ticks

    # Penalty for insufficient distance when "going" to discourage risky decisions
    if not self.collision:
        if self.distance_at_go < 3:  # Assuming a safe distance threshold of 3
            self.reward -= 5  # Penalty for risky decisions
# (continues)
```

3 Results and Conclusions

The new updated model is indeed a lot more cautious in its approach, and achieves notably fewer crashes, overall prioritizing waiting in more instances than the original models. As can be seen in Figure 1, we can see that the example critical distance of 13 is not only resolved without collision (which is difficult to achieve by the original noisy observer), but more critically the implemented safety margin is respected. The agent patiently waits the extra steps to ensure the no collision happens at a safe distance. The increased safety is corroborated by the iteration results, as they serve as a testament that crash chances have been greatly reduced (to virtually zero).



(a) The new model choses to wait for the critical distance of 13.

i	t	r	d	w	c
0	1.34	-16.36	41.24	0.0	0.18
1	1.38	-7.18	45.08	0.0	0.09
2	1.76	-13.3	42.14	0.01	0.15
3	2.64	-10.24	42.65	0.01	0.12
4	4.98	-10.24	40.18	0.07	0.12
5	8.32	-1.06	41.92	0.21	0.03
6	7.79	-1.06	40.39	0.22	0.03
7	7.54	-0.04	41.52	0.19	0.02
8	8.14	2.0	39.38	0.22	0.0
9	5.56	2.0	42.28	0.16	0.0

(b) After 10 iterations, crash chances are nearly 0.

Figure 1: Gaze tracking results for 5 different targets, with different positions and width.

Using the original tools provided during the lecture, the results can also be visualized as in Figure 2, where the prevalence of waiting is shown even more clearly; as well as the ability to avoid crashes.

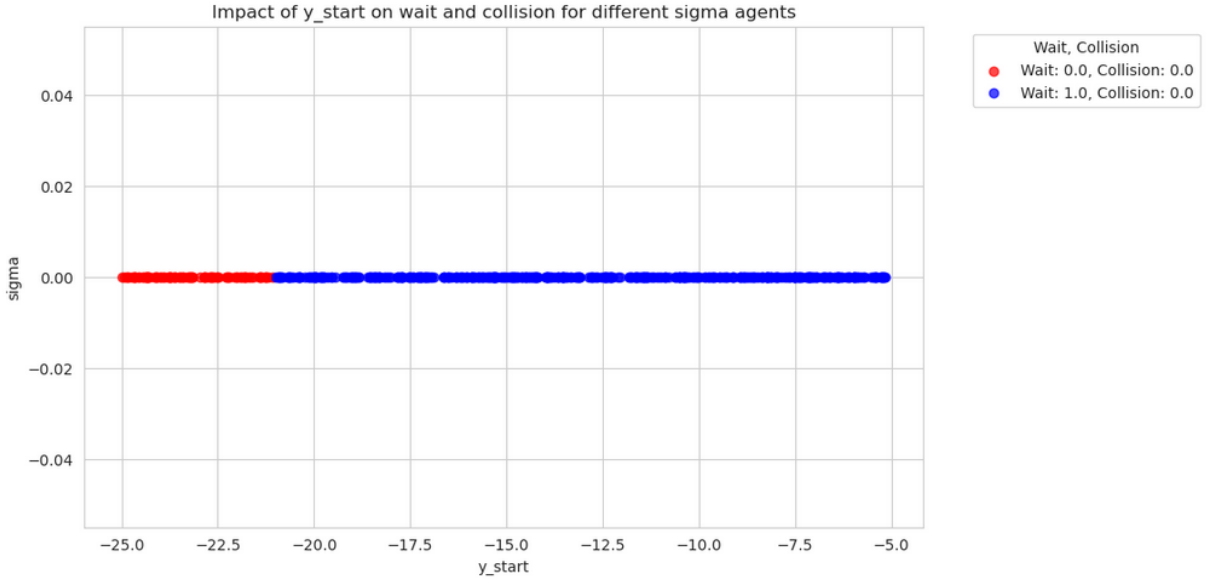


Figure 2: Go/no-go decisions for different starting y values, all resulting in no collisions.