

Laboratório VISGRAF

Instituto de Matemática Pura e Aplicada

About Idle Behaviors of Autonomous Agents

Caio Souza, Luiz Velho

Technical Report TR-21-07 Relatório Técnico

July - 2021 - Julho

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

TECHNICAL REPORT 2021.2

CAIO SOUZA*, LUIZ VELHO*

July 1, 2021

CONTENTS

1	Introduction	1
2	Overview	1
3	Idle behaviors	2
3.1	Look At Behavior	3
3.2	Wandering Behavior	4
4	Future Steps	5

1 INTRODUCTION

Our last report (2021.1) overviews the DogBot agent timeline development state up to the test scene with reactive behaviors (or behaviors that initiate from an explicitly player's command and have well-defined goals). While this type of behavior can generate various exciting applications, such as our fetch and hula hoop jump, our conception of an intelligent agent also needs another kind of behavior to mimic our real world. These behaviors are the ones triggered by the agent itself.

Here we will address precisely this facet of our agent's own will and present our current agent's implementation organization diagram. We will introduce two new behaviors under the *Idle Behavior* umbrella; the first is wandering and the second is look at (which is driving our agent "look" at something catching its attention). We believe these two behaviors can add another depth level to our agent and make it more dynamic to its environment other than just interacting with the player.

2 OVERVIEW

The block diagram illustrates our current Dogbot agent organization. It follows our proposed three-level hierarchy, which uses Reinforcement Learning as part of the mid-level controller in conjunction with other traditional controllers.

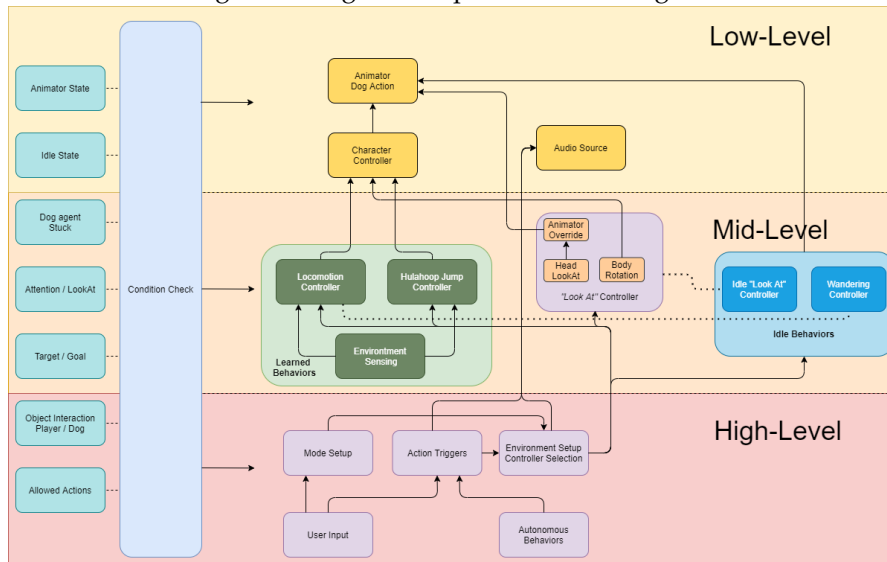
One crucial component of this system is a long block "Condition Check" extending over the hierarchy with a few examples attached to its left. This block consists of our game logic and state machines present in different parts of the stack and is specific to our application.

HIGH LEVEL The first level of control from which every action initiates. Two essential blocks are:

- User input: this accounts for the actions initiated by the player. It can be any input, varying from traditional mouse and keyboard to Virtual Reality tracking, voice commands, etc. These inputs are directly related to our reactive behaviors, where the agent responds to a stimulus.

*Instituto Nacional de Matemática Pura e Aplicada, Estrada Dona Castorina 110, 22460-320 Rio de Janeiro - RJ, Brazil.

Figure 1: DogBot's implementation diagram



- Agent autonomous behaviors: this block represents the source of its own agent will. Our current implementation only controls the agent's idle behaviors, but it is not limited to it. One example where it applies in other situations could be when the player calls the agent, DogBot can or not answer based on its affinity with the player.

The environment is set up from these two sources, and the appropriate messages are passed down to the Mid-Level.

MID-LEVEL specific tasks and can be used simultaneously to create more complex behaviors (for example, when you play fetch, you can use the locomotion together with the head movement to track the stick thrown by the player).

Here, the first group comprises the learned behaviors using reinforcement learning and being responsible for locomotion and dexterous moves (i.e., the hula hoop jump). Next, the deterministic controller for the head/body positioning of the dog's "look at".

Finally, the idle behaviors function in a stochastic way, handling the *wandering* and *look at* activation, notice that they also can use functionalities from the other controllers for locomotion and look at together with their specifics.

LOW-LEVEL The bottom level of our hierarchic approach is the closest to the game engine, comprising the animation generation, visualization, and audio. Interestingly, the character controller and Animator have their proper state machines and logic, but these details are not visible to the higher levels. This separation forces the top levels to achieve the desired task/goal with the limited tools we allow them to and ensures that the final animation result is inside the permitted animation set, which is a desired property for controlled environments.

3 IDLE BEHAVIORS

Dogbot's idle controller is responsibly selecting and updating the behavior given the game logic and environment conditions. This controller runs simultaneously in the background with the agent control modes and represents the agent's will when no actions/interactions with the player are in progress. Currently, there are two behaviors available: Look At and Wandering, with their selection and functionality based on stochastic procedures.

While being on the mid-level like the other controllers, the idle controller runs in a slower time-scale (1 Hz) than the other controllers (30/60 Hz) and can use the other controllers to complete their tasks. It is also a low priority controller, which meant its tasks could be canceled or overridden by the reactive tasks at any point. These properties fit well within the idle controller intent; being in a coarser time-scale allows it to stay on a higher control level and delegate tasks to other controllers. Having a low priority does not delay or affect the handling of other inputs and behaviors.

Our implementation of the idle system bases in three components for each behavior:

- pre-conditions: Game and environment state needed to enable the behavior.
- probability: The change of the behavior occurring once the preconditions are met.
- cooldown time: Once the behavior has been activated, the delay needed to re-activate it.

These three enabled us to models complex behaviors more organized and straightforward.

3.1 *Look At Behavior*

Figure 2: Look At turning both the dog head and body to face the player.

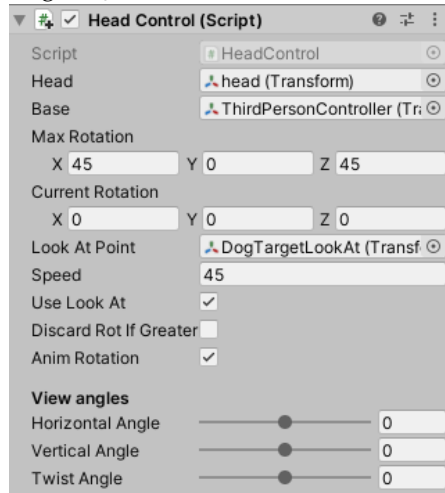


The "Look At" behavior intends to add dynamics to the dog's interaction and its environment. Its first design works by the pre-condition of something calling the dog's attention, for example, the player moving closer or any event happening close. This behavior outcome is the dog turning its neck together with its body if required and look at the thing which called its attention for a random amount of time.

Our implementation of the dog head control is entirely deterministic and procedural. A common approach to head animation is using layer masks and playing different animations for each part of the body. Nevertheless, our dog does not have separated animations, and hence we opted for going the procedural route.

Inside Unity, each dog's bones are represented by nested GameObjects. The Animator individually positions them at each time step according to its character controller blend trees and state machines. We use a C# script to control the head. Its key detail is changing the head positioning inside "LateFixedUpdate" to override the Animator work, which happens in the "FixedUpdate" time tick.

Figure 3: Parametric head controller



Our head controls are the Horizontal, Vertical and Twist angles, which vary between $[-1, 1]$ and are later mapped to the user-defined bounds. Note that these angles are applied to the head bone and are relative to the root bone, not the head parent bone, to base our calculations in the dog body forward vector without being affected by other bones changes.

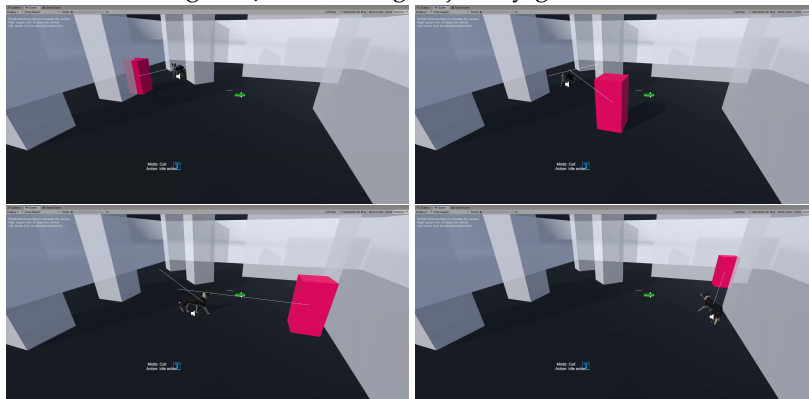
Lastly, there are two helpers to facilitate the usage. One of them is the possibility of setting an ending configuration for the angles and automatic animating from the current; its only parameter is the speed to animate. Next, there is the possibility of setting a reference point (Transform) to look at, and the C# script will handle all calculations turning both the body and head to point to the desired location.

One caveat of this functionality is that it calculates the angles by projecting the point in the frontal vision plane of the dog, similar to a camera projection, given that it can only compute the Horizontal and Vertical angles.

3.2 Wandering Behavior

The wandering behavior seeks to break the monotony when there is no activity. In these situations, if the dog stays still for too long, it can go unnoticed and be overshadowed by the environment, especially in Virtual Reality, where the user can be amazed by exploring the environment. For this reason, adding such autonomous behavior help to acquire the user's attention and makes the dog more noticeable.

Figure 4: Wandering trajectory generation

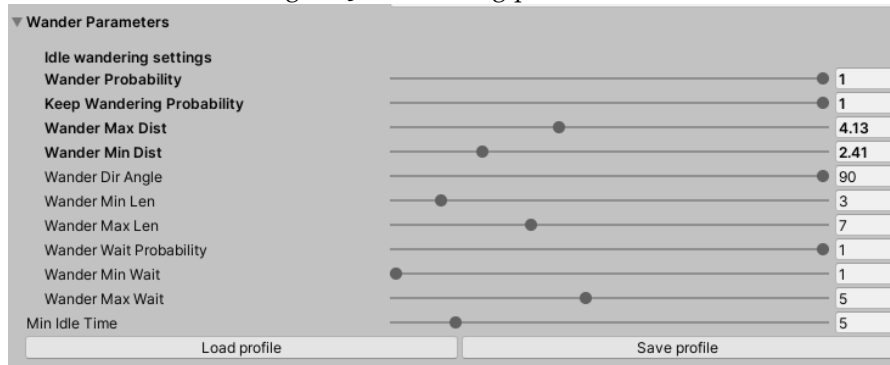


This behavior combines both the learned movement controller with randomly generated trajectories. We greedily construct these trajectories by choosing random directions inside the agent view cone and walking random

lengths. To make it more befitting, the agent can also stop for random amounts of time and look around from time to time.

The trajectory patterns are parametric and can be changed at any time. There is an option to save and load these parameters under profiles, simplifying the setup of the generator.

Figure 5: Wandering parameters



One drawback noticed by greedily generating trajectories was the strange flickering of the agent between turning left and right. This behavior was more pronounced when its destination target was close, and its speed was low, the exact case of the wandering behavior. We believe this is due to the lack of temporal coherence of the neural network output aligned with the high decision rate of the agent (30 Hz). Decreasing the decision rate would turn the agent unresponsive and unable to complete dexterous tasks, i.e., the hula hoop jump, so the solution was to apply temporal smoothing to the actions.

Our smoothing choice was the exponential moving average over the continuous output actions; X-Axis movement forward and backward, and the Y-Axis turning left and right. While the network also outputs continuous values for the crouch and jump actions, they work on thresholds and are equivalent to discrete actions. Therefore, if we apply any smoothing to it, the same unresponsiveness problem can occur in the jump task.

4 FUTURE STEPS

Our future steps include experimenting with behaviors in-between the reactive and idle, where there is neither an explicit user command nor an idle action, but a free interaction of the dog and the player, where both sides can trigger it. These interactions are the last challenge to complete the tool-set to make our agent intelligent in our proposed abstraction. While the amount and complexity of behaviors are small, they work as proof of concept and paves the road for future progress.