Self-organized collective escape in bird flocks

Papadopoulou, Marina

DOI:
10.33612/diss.255711610

Chapter 6.

DaNCES: a framework for data-inspired agent-based models of collective escape

~ M. Papadopoulou, H. Hildenbrandt, & C.K. Hemelrijk ~

Abstract

Our understanding of emergent phenomena during collective escape in animals has been limited by the lack of empirical data and the complexity of patterns of collective escape seen in nature. Different species may react differently to an approaching predator with individuals choosing their reaction from a large repertoire of evasive maneuvers. Developing a model that captures this large variety of individual escape reactions and reproduces patterns of collective escape seen across species is challenging. To tackle the complexity and diversity of patterns, here, we developed a framework that uses individual-based state machines to model collective escape. In our flocking models, an individual, based on its state machine, can switch its behavior between 'flocking' states with different coordination specifics and 'escape' states with different maneuvers, depending on the predator's behavior. Similarly, a new agent-based model in our framework is easily composed by a set of states (which include rules of motion and interaction). The sequencing of these states in a model enables the simulation of long events of collective escape adjusted to empirical data. The simulations of a model in our framework can completely change through the configuration of model parameters (at run-time) and with very minimal code alterations, showing great potential for future use across species and ecological contexts.

6.1 Introduction

Since the first model to reproduce patterns of collective escape in fish schools was published 20 years ago [86], few have attempted to model patterns of collective escape across taxa (e.g., [31, 186, 74, 76]), leaving many gaps in our understanding of collective escape. Even though species differ a lot in their specifics of individual escape reactions, models of collective escape are usually generic (Vicsek-like, e.g., [186, 3, 17, 76]). Most models include a rule of predator avoidance that is always balanced with the coordination rules among individuals during collective motion ('continuous' rule), for instance a tendency to turn away from the predator's position, (e.g., [86, 186]), or an instantaneous turning motion that 'interrupts' the regular coordinated motion of the group (a 'fixed' or 'discrete' reaction, usually modeled to study wave propagation, e.g., [76, 74]). The escape reactions we see in nature may be represented by both such rules; knowledge about which type of escape reaction individuals perform when their group is under attack, however, is still lacking. Additionally, more complex escape maneuvers (e.g., protean motion [80, 91, 123]) are rarely modeled in the context of collective behavior. Thus, in order to improve our modeling of collective escape, both continuous and discrete escape reactions should be modeled at the individual level. Nevertheless, before simulating the collective escape of a specific species, a model of collective motion should first be adjusted to the species' characteristics.

The majority of agent-based models of collective behavior of animals are based on the rules of attraction, alignment and avoidance [31]. Given that species often differ in their flocking behavior (specifics of coordination, shape and density of their flocks), a model needs to be adjusted to empirical data in order to realistically simulate a flock in nature. Often, mere changes in a model's parameters are not enough to do so; adjustments in the interaction rules at the level of the individuals are needed. How exactly these coordination rules are modeled may also vary. For instance, an individual may avoid collisions by turning away from the position of its neighbor, by aligning with the heading of its neighbor, or by decelerating. How these differences affect the emerging collective motion has been rarely studied in combination with empirical data. Additionally, the majority of computational models of collective behavior assume that individuals are identical, even though individual variation in specific traits within a group has been identified in many species [89, 155].

Since interaction rules are the core of computational models of collective behavior, extensive changes are necessary to adjust a model to a new species or context. For instance, in Chapter 2 and 3, for our model to include properties of pigeon flocks [155, 140], we modeled a rule of accelerating cohesion (in addition to centroid attraction): individuals deviate from their preferred speed to stay with the flock. While modeling starlings in Chapter 5, we added a rule of roost-attraction instead, to make the flock stay close to their roost as seen

in nature [188]. In a model of jackdaws, one may have to set up mating partners and a rule for reduced topological range as found in empirical data [113]. These species-specific adjustments can be time-consuming and often challenging to make in an existing model, given the increasing model complexity as more species-specific characteristics are added. Additionally, very specialized models may be of 'single-use' [59], which may delay advances in the field and introduce disparities between different models that limit the generalization of conclusions and model comparisons.

Here, we recognized the necessity for a new modeling structure that can counteract the aforementioned issues. On the one hand, we want to create a model that is flexible and easily adjustable to different species and ecological contexts. On the other hand, it should accommodate the inclusion of several (and perhaps diverse) types of individual motion (especially escape), from continuous coordinated motion to sudden reactions to the predator. To tackle these, we developed a modeling framework (named DANCES, DAta-iNspired Collective EScape) that uses individual-based state-machines to model collective escape (transitions between flocking and escaping). Collective states and transitions between the states have been discussed in collective behavior of animals [175]. By combining internal states at the individual level with a composable model structure, we further highlight the potential of such conceptualization to study collective escape.

6.2 Modeling framework

6.2.1 Individual-based state machines

An agent in our framework is represented by its 'internal state'. In the context of collective escape, this internal state consists of the agent's position and velocity in a global coordinate system. Since agents in computational models of collective behavior move in space based on their interactions with one another, we conceptualize an individual-level 'rule' in our model that can alter the internal state of an agent (and thus control its motion), as a stand-alone 'internal-state control unit' (referred to as an *ISC-unit*). Examples of such units are an alignment interaction (that makes a focal individual align its heading with its neighbors), a roosting behavior (that makes a focal individual turn towards its roost), or a noisy motion (that adds a random error to the heading of a focal individual). Each ISC-unit owns a set of parameters (e.g., the number of interacting neighbors or the relative position of the roost). A list of all ISC-units included in the models presented in the previous chapters (*HoPE*, *ColT*, and *StarEscape*) is given in Table 6.1.

Table 6.1: **ISC-units** in the models *HoPE* (Chapter 2 and 3), *ColT* (Chapter 4) and *StarEscape* (Chapter 5). Each unit is stand-alone and could be combined with other to create states.

Name	Description	Model
Prey		
align_n	Align with a number of closest neighbors.	*HoPE*, *ColT*, *StarEscape*
cohere_turn_n_all	Turn towards the average position of a number of closest neighbors.	*HoPE*
cohere_centroid_distance	Turn towards the average position of a number of closest neighbors depending on the agent's distance from it (the closer the average position the weaker the attraction).	*ColT*, *StarEscape*
cohere_accel_n_front	Accelerate towards a number of closest neighbors within a frontal field of view. Decelerate if no neighbors are within the field of view.	*HoPE*
avoid_n_position	Turn away from the average position of a number of closest neighbors if it is within a radius of minimum separation.	*HoPE*, *ColT*, *StarEscape*
avoid_p_direction	Turn away from the heading of a close-by predator.	*HoPE*
random_t_turn_gamma_pred	Turn with an angular velocity sampled from a gamma distribution, with direction away from the predator.	*HoPE*, *ColT*
copy_escape	Copy the state of a close-by neighbor if the state is copyable.	*ColT*, *StarEscape*
relative_roosting_persistant	Turn towards a give point in space relative to the flock's center.	*ColT*
roost_attraction	Turn towards a give point in space, if the agent is outside a given radius around the point.	*StarEscape*
altitude_attraction	Pitch towards a preferred altitude if the agent is outside a given altitude range (3D).	*StarEscape*

Table 6.1: **Continued**

Name	Description	Model
dive	Pitch downwards (dive) away from a close-by predator (3D).	*StarEscape*
Predator		
set_retreat	Reposition at a given position relative to the flock.	*HoPE*, *ColT*, *StarEscape*
select_flock	Choose a flock as a target.	*HoPE*, *ColT*, *StarEscape*
shadowing	Follow the target from a given bearing angle and distance.	*HoPE*, *ColT*
chase_closest_prey	Move towards the closest prey (at every instance) with a speed scaling from the target's speed.	*HoPE*, *ColT*
lock_on_closest_prey	Move towards a targeted prey (constant during the state) with a speed scaling from the target's speed.	*HoPE*, *StarEscape*
avoid_closest_prey	Turn away from the position of the closest prey.	*HoPE*, *ColT*
hold_current	Circle around a given position.	*HoPE*, *ColT*, *StarEscape*
position_to_attack	Position at a given distance and bearing angle in the horizontal and vertical plane from the target (3D).	*StarEscape*
General		
wiggle	Turn by a random angle (noise).	*HoPE*, *ColT*, *StarEscape*

We treat each internal-state control unit as a building block; the modeler combines the pieces that match his/hers/their study system and behavior of interest. A set of ISC-units defines a 'state'. For instance, a state that represents a 'flocking' behavior may consist of four ISC-units that affect the agent's behavior while moving: alignment, centroid-attraction, avoidance and noise. Each state changes the value of a variable that controls the agent's motion. Thus, an ISC-unit is designed around the nature of its effect on individual motion. For instance, as given in our previous models' descriptions, in our models this

variable is a steering vector (Eqs. 2.4, 2.2, 3.27, 4.1). Each ISC-unit adds to the main steering vector of a state, which in turn affects the internal state of the agent. The use of such steering vector fits in our flocking models, but it is not enforced by our framework. Apart from the parameters of each ISC-unit of a state, a state also has its own parameterization. This can include, for instance, the frequency with which the steering vector is updated (recalculating the effect of each ISC-unit, also known as 'reaction frequency'). Overall, a model with a single state in our framework is the equivalent of a classic model of collective behavior, in which a constant set of rules at the individual level affects the motion of all agents identically.

To model discrete reactions and switches between different behaviors (related to the predator or the ecological context), our framework enables several states to be included in a model. An agent is thus defined by not only its internal state, but also a finite-state machine that controls its behavior. Each state can comprise a different combination of ISC-units or the same ISC-units with different parameterization. For instance, a state of 'flocking' can be followed by an 'escape' state, during which an agent flocks while also avoiding a predator. An example of the definition of a state machine of two different agents is given in Examples 6.1 and 6.2.

Example 6.1: Example of a definition of an agent with a single state to align with, be attracted to, and avoid its close-by neighbors.

```
// States and internal control units composition of a simple agent
using AP = states::package<
     states::transient<
        iscus::package<DummyAgent,
            iscus::align<DummyAgent>, // alignment
            iscus::cohere<DummyAgent>, // cohesion
            iscus::avoid<DummyAgent> // avoidance
      >>
    >;
```

Example 6.2: Example of a definition of a state machine of a 'prey' agent (*PreyType*) with several states. A description of each ISC-unit is given in Table 6.1 and their parameters across states are given in Example 6.3.

```
// States and internal control units composition of a PreyType agent
using AP = states::package<
     states::transient< // 1.regular flocking
        iscus::package<PreyType,
            iscus::align_n<PreyType>,
            iscus::cohere_centroid_distance<PreyType>,
            iscus::avoid_n_position<PreyType>,
            iscus::roost_attraction<PreyType>,
            iscus::wiggle<PreyType>
      >>,
      states::transient< // 2. alarmed flocking
        iscus::package<PreyType,
            iscus::align_n<PreyType>,
            iscus::cohere_centroid_distance<PreyType>,
            iscus::avoid_n_position<PreyType>,
            iscus::roost_attraction<PreyType>,
            iscus::copy_escape<PreyType>, // with copy escape
            iscus::wiggle<PreyType>
```

```
19        >>,
20        flee_state, // 3. escape multi-state
21        states::persistent< // 4. refraction - alarmed flocking
22          iscus::package<PreyType,
23              iscus::align_n<PreyType>,
24              iscus::cohere_centroid_distance<PreyType>,
25              iscus::avoid_n_position<PreyType>,
26              iscus::roost_attraction<PreyType>,
27              iscus::wiggle<PreyType>
28        >>
29      >;
30
31  using flee_state = states::multi_state<PreyType,
32        states::persistent< // 1.escape turn
33          iscus::package<PreyType,
34              iscus::random_t_turn_gamma_pred<PreyType>,
35              iscus::align_n<PreyType>,
36              iscus::cohere_centroid_distance<PreyType>,
37              iscus::avoid_n_position<PreyType>,
38              iscus::wiggle<PreyType>
39        >>,
40        states::persistent< // 2.uncoordinated dive
41          iscus::package<PreyType,
42              iscus::dive<PreyType>,
43              iscus::wiggle<PreyType>
44        >>
45      >;
```

Transitions between states are controlled by a transition machine, owned by each agent. Each state has a duration that ranges from a single reaction step to any parameterized duration. At the exit of a state, the transition machine assigns the next state as a function of the previous state combined with a set of probabilities. These probabilities can be predefined through parameterization (resembling a Markov-chain). However, some transitions may depend on specific conditions during a simulation. For instance, an individual may switch to an escape state only if the predator is approaching. Thus, each transition machine also owns a variable that can alter the probability of switching to a specific state. We used such transition-altering mechanism through our 'alertness' variable (in the *HoPE* and *StarEscape* model): the closer the predator, the higher the probability to perform an escape maneuver. On the contrary, we used a completely deterministic loop for the behavior of the predator in our models (the predator has a probability of 1 to transition between its 'pursuit', 'attack' and 'retreat' states that all have a pre-defined duration, e.g., Figure 5.1c).

To account for cases where the state selection should be performed by a specific ISC-unit and not the transition machine, we further included 'multi-states' in our framework. A multi-state is a mere collection of states. The states of a multi-state (sub-states) have a conceptual connection that affects their role during a simulation. For instance, the two escape maneuvers of *StarEscape* (dive and turn) belong to a multi-state of escape. The transition machine may decide that an individual should perform an escape maneuver, but the probability of selecting each one can be altered by each ISC-unit during the simulation (for instance if the predator attacks from above, a diving maneuver may be evaluated as more effective), or from the parameterization of the multi-state

(for instance, empirical data may show that the frequency of turning is higher than the frequency of diving). An example of the definition and parameterization of a multi-state is given in Examples 6.2 ('flee_state') and 6.3, respectively.

6.2.2 Multi-level parameterization

As a result of the structure of our framework, parameters in a model can be at the level of the agent (such as the mass of an agent), at the level of a state (for instance a state-specific speed), and at the level of an ISC-unit within a specific state (such as the number of topological neighbors for alignment). To control all these, a configuration file with all the necessary parameters should be given as input in the model (at run time). We implemented this as a JSON file given its nested nature that allows for parameterization of very complex states machine. We can thus have control of every detail of our model. This is particularly helpful when running a large set of simulations with different parameterization: some ISC-units can just be activated or deactivated and thus completely change the model between runs. An example of a configuration file is given in Example 6.3.

Example 6.3: Example of a configuration file (JSON type) for the parameterization of a prey agent with the state definition given in Example 6.2. The order of the ISC-units in the configuration file matters; it should match their order in the state definition. The file is given as input to the compiled model at run time.

```
{
  "PreyType": {
   "N": 100,
   "states":
   [
    {
     "name": "transient",
     "description": "regular
       flocking",
     "tr": 0.05,
     "iscus":
     [
      {
       "name": "align_n",
       "topo": 7,
       "fov": 270,
       "maxdist": 200,
       "w": 0.5
      },
      {
       "name":
         "cohere_centroid_distance",
       "topo": 7,
       "fov": 270,
       "maxdist": 200,
       "min_w_dist": 0,
       "max_w_dist": 5.0,
       "w": 1
      },
      {
       "name": "avoid_n_position",
       "topo": 1,
       "fov": 270,
       "minsep": 0.8,
       "maxdist": 200,
       "w": 0.5
      },
      {
       "name": "roost_attraction",
       "roost_radius": 100,
       "roost_pos_xz": [ 50, 100 ],
       "w": 0.25
      },
      {
       "name": "wiggle",
       "w": 0.1
      }
     ]
    },
    {
     "name": "transient",
     "description": "alarmed
       flocking",
     "tr": 0.02,
     "iscus":
     [
      {
       "name": "align_n",
       "topo": 7,
       "fov": 270,
       "maxdist": 200,
```

```
      "w": 0.5
     },
     {
      "name":
        "cohere_centroid_distance",
      "topo": 7,
      "fov": 270,
      "maxdist": 200,
      "min_w_dist": 0,
      "max_w_dist": 5.0,
      "w": 1
     },
     {
      "name": "avoid_n_position",
      "topo": 1,
      "fov": 270,
      "minsep": 0.8,
      "maxdist": 200,
      "w": 0.5
     },
     {
      "name": "roost_attraction",
      "roost_radius": 100,
      "roost_pos_xz": [ 50, 100 ],
      "w": 0.25
     },
     {
      "name": "copy_escape",
      "topo": 7,
      "fov": 270,
      "maxdist": 200
     },
     {
      "name": "wiggle",
      "w": 0.1
     }
    ]
   },
   {
    "name": "multi_state",
    "description": "flee state",
    "copyable": true,
    "selector": {
      "probs": [ 10, 0, 0, 0 ],
      "override_from_iscus" : true
     },
    "sub_states": [
     {
      "name": "persistent",
      "description": "escape turn",
      "copyable": true,
      "tr": 0.02,
      "duration": 2,
      "iscus":
      [
       {
        "name":
          "random_t_turn_gamma_pred",
        "turn_mean": 180,
        "turn_sd": 5,
        "time_mean": 2,
        "time_sd": 0.1,
        "select_prob": 0.5
       },
       {
        "name": "align_n",
        "topo": 7,
        "fov": 270,
        "maxdist": 200,
        "w": 0.5
       },
       {
        "name":
          "cohere_centroid_distance",
        "topo": 7,
        "fov": 270,
        "maxdist": 200,
        "min_w_dist": 0,
        "max_w_dist": 5.0,
        "w": 1
       },
       {
        "name": "avoid_n_position",
        "topo": 1,
        "fov": 270,
        "minsep": 0.8,
        "maxdist": 200,
        "w": 0.5
       },
       {
        "name": "wiggle",
        "w": 0.1
       }
      ]
    },
    {
     "name": "persistent",
     "description": "uncoordinated
       dive",
     "copyable": true,
      "tr": 0.02,
       "duration": 3,
       "iscus":
       [
        {
         "name": "dive",
         "minsep": 20,
         "w": 0.5,
         "max_dive": 4,
         "select_prob": 0.5
        },
        {
         "name": "wiggle",
         "w": 0.1
        }
       ]
      }
     ]
    },
    {
     "name": "persistent",
     "description": "escape penalty -
       alarmed flocking",
     "tr": 0.02,
     "duration": 4,
     "iscus":
     [
      {
       "name": "align_n",
       "topo": 7,
       "fov": 270,
       "maxdist": 200,
       "w": 0.5
      },
      {
       "name":
```

```
          "cohere_centroid_distance",
        "topo": 7,
        "fov": 270,
        "maxdist": 200,
        "min_w_dist": 0,
        "max_w_dist": 5.0,
        "w": 1
       },
       {
        "name": "avoid_n_position",
        "topo": 1,
        "fov": 270,
        "minsep": 0.8,
        "maxdist": 200,
        "w": 0.5
       },
       {
        "name": "roost_attraction",
        "roost_radius": 100,
        "roost_pos_xz": [ 50, 100 ],
        "w": 0.25
       },
       {
        "name": "wiggle",
        "w": 0.1
       }
      ]
     }
    ],
```

```
    "transitions": {
     "name":
       "piecewise_linear_interpolator",
     "TM":
     [
       [
         [ 1, 0, 0, 0 ],
         [ 1, 0, 0, 0 ],
         [ 0, 0, 0, 1 ],
         [ 0, 1, 0, 0 ]
       ],
       [
         [ 0, 1, 0, 0 ],
         [ 0, 0.999, 0.001, 0 ],
         [ 0, 0, 0, 1 ],
         [ 0, 1, 0, 0 ]
       ],
       [
         [ 0, 1, 0, 0 ],
         [ 0, 0.99, 0.01, 0 ],
         [ 0, 0, 0, 1 ],
         [ 0, 1, 0, 0 ]
       ]
      ],
      "edges": [ 0, 0.3, 1 ]
     }
   }
}
```

Our implementation of the parameterization of the transition machine is inspired by behavioral chains found in bird flocks [166], so that transition probabilities can be easily informed by empirical data. We input a Markov chain in the form of a transition matrix in our configuration file. Given that these probabilities may need to be adjusted during a simulation, depending on external conditions such as the proximity of an individual to the predator, we implemented an interpolator between several transition matrices. In detail, by giving the model several matrices (3 by default, see Example 6.3) for some extreme conditions, the exact transition matrix is created at every state-switching step based on the parameter of the transition machine. For instance, the interpolator could be used when the probability of an individual to perform a very extreme escape maneuver should be 0 when the predator is not present and only increase slightly when the predator gets very close. In this example, the transition parameter would be the distance to the predator, and we would need at least 2 transition matrices as input, one for low proximity situations and one for high. For the opposite case, when someone wants to model closed behavioral loops with agents changing their behavior deterministically, the probabilities of the required transitions can be set to 1 and all the others to 0.

Finally, the ISC-units are not specific to a type of the agent. On the contrary, they can be used by any agent as long as all its necessary variables (parameters) are there. For instance, predator and prey can use the same ISC units to move around, but to use the ISC-unit that allows an agent to select a target from a group (conceptually a predatory behavior), the agent needs to own a 'target'

variable. This flexibility allows for all of the ISC-units in our framework to be easily reusable, even when introducing a completely new type of agent.

6.2.3 Case studies: *HoPE* vs *StarEscape*

In Chapter 2, *HoPE* is a model that consists of a single state with 5 ISC-units: alignment, centroid-attraction, speeding attraction, noise, and predator avoidance. Since we model predator avoidance as a continuous tendency to turn away from the predator while coordinating, an extra state to model escape was not needed. In Chapter 3, we conceptualize collective escape to be initialized by one individual that stops coordinating with its neighbors and maneuvers to escape the predator for a specific time period. Additionally, after the escape maneuver is completed, we add a refractory time period during which an individual does not coordinate with each neighbors (to ensure that the emergence of splitting and collective turning is the effect of the group's 'decision'). Thus, the extended *HoPE* model consists of three states: a flocking state (the one from its previous version in Chapter 2), an escaping state (with two ISC-units, escape maneuver and noise), and a refraction state (with a single ISC-unit that adds noise to the straight heading of the initiator).

In *StarEscape*, the complexity of our model increases. We model 4 states: flocking, alarmed flocking, escape (which as mentioned earlier is a multi-state), and refraction. Flocking includes the following 6 ISC-units: alignment, centroid-attraction (distance biased), avoidance, noise, roost attraction, and altitude attraction. The alarmed flocking state is structurally identical with the flocking state, apart from being parameterized with a higher reaction frequency and an extra ISC-unit that affects the transition machine: a 'copy-escape' behavior. This ISC-unit checks whether any neighbor is in the escape state. If so, it sets the future state in the transition machine to be the same as its neighbors. The escape multi-state has two states that include the coordination-related ISC-units of the flocking states, and an additional maneuvering ISC-unit, one for the diving maneuver and one for the turning maneuver. Finally, the refraction state, has the coordination-related ISC-units of the flocking states but a predefined duration. Having these states separately ensures that individuals will not keep reacting to the predator in an unrealistic manner (exiting the escape state and immediately re-entering it again).

6.2.4 Customizable locomotion type

The type of motion of grouping individuals (e.g., flying, swimming, or walking) is crucial to the emerging collective behavior. For our framework to be extendable to non-flying species, the way that a steering vector is affecting the motion of an agent is controlled by a stand-alone motion integrator. This integrator functions irrelevantly of the specifics of an agent (its variables and states). This allows the adjustment of the core of a model to other study systems, without

changes being necessary on the structure of the agents and the ISC-units.

6.2.5 Real-time visualization & data analysis

To incorporate analysis and visualization in our simulations we use the concept of 'observers' (a software design pattern mainly used in event-driven computing [52]). Observers can access variables of a simulation in real time without interfering with the model. Based on a chain of specialized observers, we can perform data analysis on the simulated trajectories in real time (for instance calculating the average nearest neighbor distance or the neighbor stability). With a separate observer per analysis type, our simulations can output analyzed data along with raw trajectories. This, for instance, was extremely valuable in Chapter 4, given that measurements of neighbor stability and deviations from parallel-paths and equal-radii turns are computationally very demanding. Additionally, the observers enable real-time visualization of a simulation (for visual debugging and calibration) that can be turned-off in order to speed up the simulations, for instance when running a large number of simulations during data collection. Finally, a graphic user interface can easily be added to a model, as we did in StarEscape (Chapter 5). An example of such observer-based properties of our simulations are given in Figure 6.1.

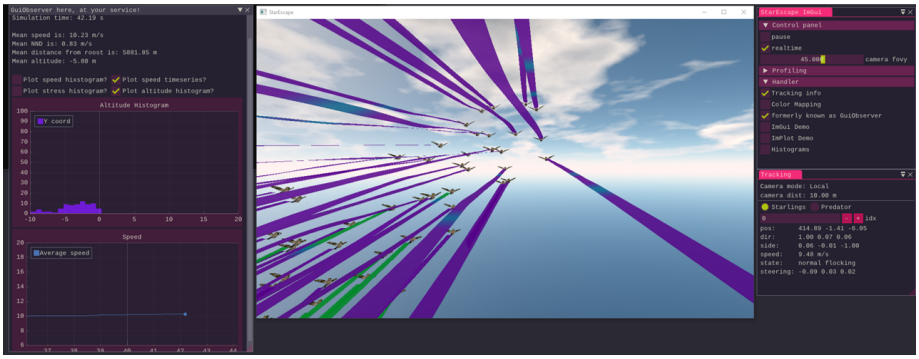

Figure 6.1: Screenshot of the GUI (created in *Dear ImGui*) and visualization (in OpenGL) of the *StarEscape* model in real time. Information from the simulation is collected through *Observers* and thus the visualization and data analysis do not interfere with the model itself, ensuring that the model can be run without them (headless version) and the code complexity does not increase.

6.3 Summary and potential

In our new framework, a model is composable, flexible and reusable. We provide a schematic overview in Figure 6.2. By the user changing a few lines of code in the definition of an agent's state machine and the configuration file, a new model can be created. A large set of already available ISC-units is given in Table 6.1. Furthermore, given the stand-alone nature of ISC-units, adding a

new one requires minimal coding and does not interfere with the chain of the model itself. Apart from adding new rules, the user can also easily remove elements of a model and quickly compare models of decreasing complexity. This is extremely valuable when searching for a simple self-organized process to explain the emergence of a collective pattern, or in order to ensure that a model falls within the 'Medawar' zone of pattern-oriented models (looking for the perfect balance between a model's complexity and the data it aims to explain [59]). Each state can be treated as an independent model.

Apart from the flexibility of our models, with the structure of our framework we can model different behaviors that are crucial to collective escape, for instance the interplay between continuous and instantaneous reactions. Because of its 'building-blocks' structure, we can develop complex models that reproduce long sequences of collective escape as the ones seen in nature without increasing the complexity of the code itself. This reduced code complexity also includes the avoidance of long series of nested conditional blocks (if-statements) which, as a model grows, become very error prone, difficult to work with, and introduce optimization issues (code that is rarely used increases the computation time). Every block in our framework (being an ISC-unit, a state, a transition machine or a motion interpolator) is a pure function that owns a set of parameters and effectively alters the state of an agent. Because of their simplicity, they are highly reusable, can be more easily optimized, while providing code clarity and debugging efficiency. Thus, the extension and growth of a model based on our framework is less error prone and less time consuming.

To conclude, our simulation framework provides a skeleton to build a new agent-based model rather than being a library. Its composable structure and detailed parameterization allows for a model to be informed with as many empirical data are available, increasing its biological relevance. Programming-wise, it does not impose our implementation, allowing a more experienced user to work in their own modeling style. Overall, if one can conceptualize a set of rules and actions that agents should follow, along with the interconnection (time sequence) of these actions, it can be modeled in our framework.

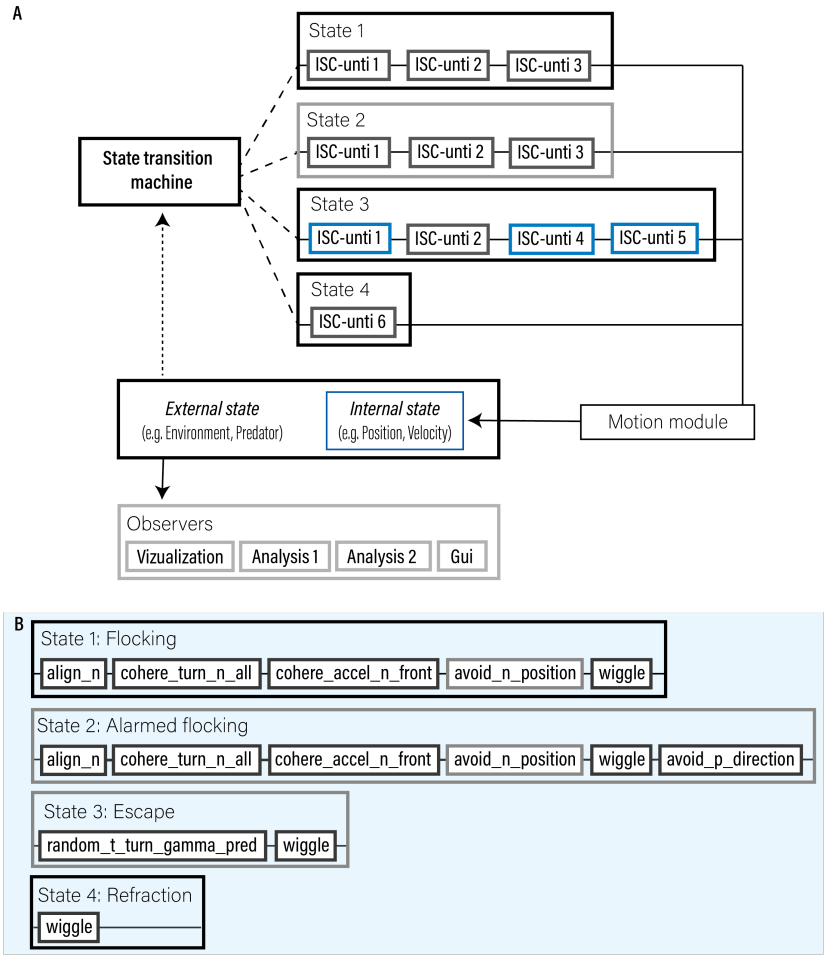

Figure 6.2: A. Schematic representation of the state machine and its connection with the other elements of our framework. The transition machine selects a state depending on the agent's internal (e.g., its position and the previously selected state), and external state (e.g., its distance to the predator). States can be composed of a different combination of ICS-units and have their own parameterization. The different border colors represent different parameter values. State 1 and 2 are identical in their ISC-units but differ in their state-level parameter values (for instance in their duration or the agent's reaction frequency). State 3 has two ISC-units in common with States 1 and 2, one of which differs in parameterization. The output of the ISC-units changes the internal state of an agent depending on the specifics of the motion module, that includes all elements related to the movement of individuals, such as the motion integrator. Observers have access to an agent's external and internal state. Several observers with different functions (e.g., to export, analyse or visualize the simulated data) can be added in a model. B. An example of the states and their ISC-units in the HoPE model (Chapter 3), relating to Figure S3.11 Figure. The *avoid_n_position* ISC-unit differs in the parameter of topological range from the other coordination related ISC-units (*align_n*, *cohere_turn_n_all*, *cohere_accel_n_front*), since we wanted flock members to avoid only their closest neighbor [72].