Master Project - Handbook

Isabel Piljek
Supervisors: Eren Cakmak, Prof. Dr. Daniel A. Keim

January 2020

1 Introduction

As the analysis of collective phenomena, often referred to as *Collective Movement*, has received much attention recently. New Visual Analytics methods to support the understanding and the analysis of spatio-temporal data in various application domains are developed. These approaches range from data aggregation of vehicle movement by spatial cells [1] to collective trends visualization in space and time by compressing the space by space-filling curves [2] or combing movement data with video to enhance sport analysis [8]. These methods need to be tested and evaluated, but although some datasets, such as three-dimensionally structured fish schools [11] or mammal movements across various species [10], exist, these usually don't cover all specifics the methods can encode and therefore these applications can't be tested thoroughly.

The overall aim of this master project was the creation of a Visual Analytics[5] prototype that creates collective movement data. The generated data must be tailored towards the capabilities of the developed techniques and must be capable of being modeled by user specifications to include motion paths and movement patterns, as described by Dodge et al. [4], as well as dealing with a modeled environment, e.g. to avoid obstacles. The resulting prototype enables the user to interactively parametrize the application itself, as well as the models, to simulate and generate collective movement that can be exported as CSV-file. It assists to tailor collective movement to desired requirements and supports diverse group formations and polarized movement within groups, also while following movement paths or performing avoidance behavior.

To create a good dataset the motion must be as realistic as possible and agents must act naturally with its environment. Sumpter[9] divides modeling strategies into mechanistic and functional approaches. Functional models aim to reflect the intention of individuals why behavior is being evolved, whereas mechanistic approaches target to explain how behavior, e.g. patterns like split and merge described by Dodge et al.[4], is achieved. Following the mechanistic strategy, two of the best-known models have been chosen as the data generation foundation.
Reynolds modeled bird-like objects[6], shortly called boid, that follow basic rules to avoid collisions with adjacent objects but adapt speed and velocity towards their neighbors to move towards their center of mass for group formation such as flocking. The boid model was extended to describe the behavior of autonomous characters[7] expressed by physical steering forces, which includes environmental behavior, such as path following and obstacle avoidance. To depict a more biologically realistic and minimalistic approach, the motion of collectives are described in a rule-based zonal model introduced by Couzin et al.[3], hereafter referred to as Couzin or zonal model. Based on the zones, movers avoid collisions within the repulsion zone by keeping a minimum distance towards others or objects that resemble their motion to align within others in the zone of alignment and inclining each other in the zone of attraction.

This handbook describes the prototype usage in section 2 and illustrates its handling by illustrating a use case. Section 3 explains in detail the parameter space of the above-mentioned models and their principles. In section 4, the setup of the application is depicted and pictures the deployment, as well as further development settings, to extend the application.

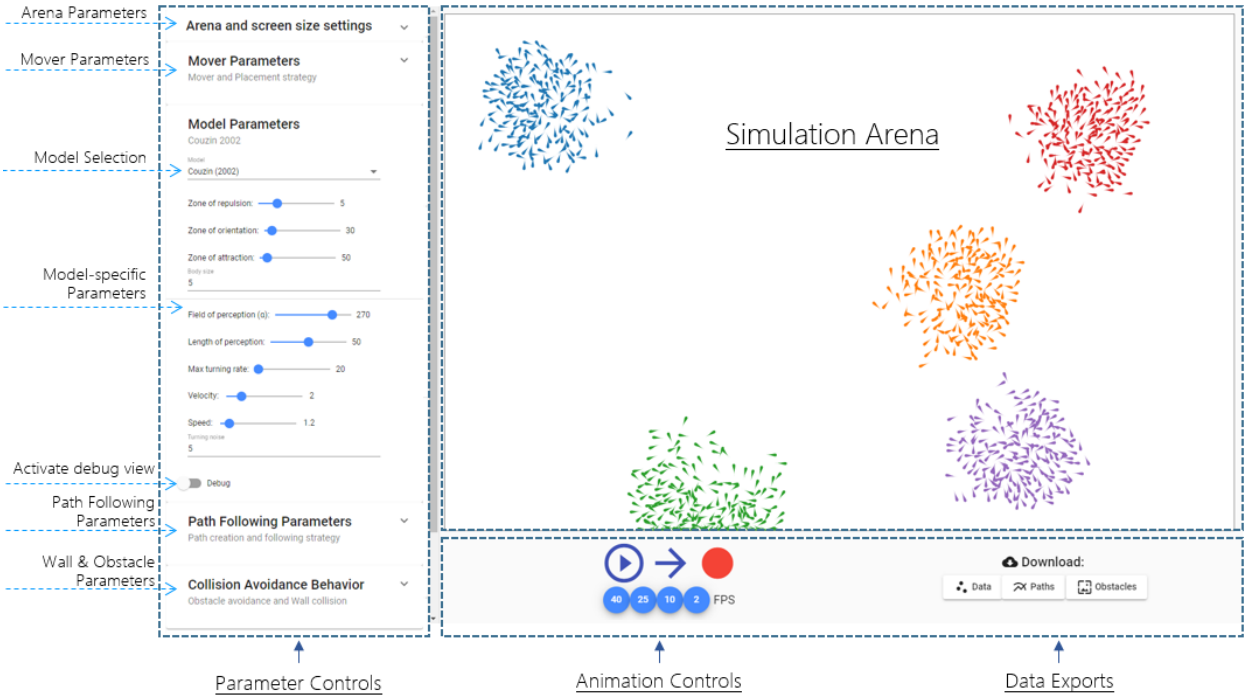


Figure 1: Overview of prototype: Parameters are controlled at the left and the resulting data is represented in simulation arena. Generated movement can be either animated or stepwise explored.

2 Prototpye usage

The developed application consists of three main parts, see figure 1: On the left side, program and model properties are grouped in the **Parameter Controls**. Through interactivity, these parameters can be changed and affect the movement behavior, which is visually displayed in the **Simulation Arena**. The simulation, as well as the **Data Exports** of the generated data, is controlled by the lower component that also contains the **Animation Controls** to animate the generated data or to explore it stepwise.
This section demonstrates and explains the various usage and handling of the prototype and starts with illustrating a basic use case. Afterward, the general parameters and their functionality are explained, followed by the model-specific movement generation. The global and model-related properties are summarized and described in more detail in section 3. As the Couzin model assumes an empty environment, path following and obstacle avoidance have only been implemented in the Reynolds model, as the extension of the zonal model is beyond the scope of this master's project.
Use Case - Exemplary demonstration: To illustrate the usability of the application, imagine an ecologist wants to generate data that includes a milling pattern, for instance. To do so, the user first adjusts parameters in the parameter controls, as the size of the arena, then the number of agents is defined that are randomly distributed in the simulation arena. The user selects the Couzin model and causes the animation in the animation control by clicking on the play button. With the configured parameters, the pattern has not been accomplished yet, so the user adapts the model properties further to his obligations. To comprehend the behavior, the debug option is enabled and the user sees in which zones movers interact with others to get an impression of the modeled behavior. By adapting the zones, the hitherto randomly placed movers form small groups. Independently from the zonal concept, movers repel and the formerly established group splits up. To avoid this undesired behavior, the field of vision and the turning rate is investigated and the user apprehends, that the turning rate was chosen to high. After reducing the turning rate, the groups perform a polarized movement. Decreasing as well as the zone of orientation, the groups now start to produces the

desired milling pattern. Once satisfied with the parametrization, the user records the data in the animation controls, stops it approximately after 2 minutes and downloads the data set.
Realizing that the generated data contains small groups, and not all members perform the desired milling pattern, the user decides to choose the Reynolds model with a preset circular motion path, that forces the movers to follow. The model is switched and the Reynolds model parameters are loaded in the parameter controls. The user activates path following and chooses the best-suited path and adapts the path-following properties further to the aspired demands. The simulation is triggered afresh and observed.

General parameters and functionality

A distinction can be made between general and model-specific parameters. General parameters control the size of the arena or the number of agents and their initial placement within the arena whereas model-specific parameters define the movement behavior based on the selected behavioral model.
Arena and screen size settings: As shown in figure 2a), the arena size can be adjusted by the user. The arena width and height can be set in a general user-specific coordinate system which is specified in units. Units can correspond to millimeters, miles, feet or other numeric units. All calculations are applied in arena units. Based on the available screen space, the corresponding pixel size is calculated proportionally so that the aspect ratio is maintained.

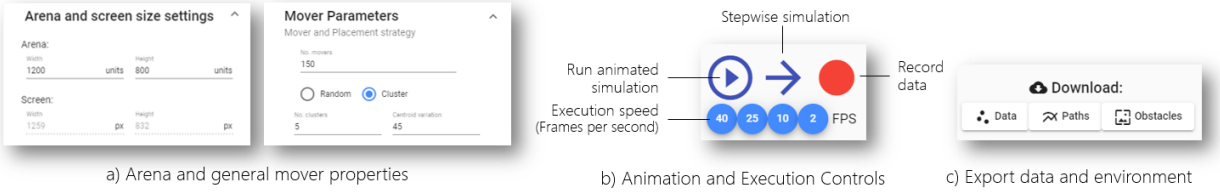


Figure 2: a) General parameter groups: Arena and Mover properties b) Functionality of animation and execution panel. c) Data export controls

Mover parameters: The number of movers and their initial placement within the arena can also be defined, see figure 2a). The agents are either distributed randomly throughout the arena (3a)) or clusters can be defined. The cluster centers are randomly distributed in the arena space with a global group orientation ranging from 0 to 360°. The deviation to this orientation can be controlled by a parameter `Centroid variation`. This corresponds to the group alignment +- random({0, variation}). In figure 3b) two clusters with a variation of 45° and in figure 3c) five groups with a variation of 10° from the group orientation are displayed. By parametrization, the user can influence the initial placement and the group alignment, as diffuse orientations within a group can lead to a split of the group.

Animation Controls: Once the agents are placed in the arena, the simulation can be run with the pre-configured or changed model-specific parameters, see figure 2b). The animation is run by clicking on the play button and can be paused with the same button. The animation and execution speed is controlled by the frame rate. The frame rate defines, how often the movement steps are executed within one second. So choosing 40fps means, the agents move 40 times per second. When debugging and *slow motion* is desired, the user either chooses a small frame rate or goes step-wise through the movements. Before recording, the simulation has to be stopped. The recording is initiated by clicking the record-button and stops vice versa.

Data Exports: The record button triggers the execution of the movement steps and saves them in the background. When the recording is stopped, the generated data can be downloaded by clicking on the `Data` button, see figure 2c). To reproduce the environment, environmental settings, such as paths and obstacle positions and offsets, can be downloaded as well by clicking on the respective buttons (`Paths` and `Obstacles`).

Figure 3: Different mover placement strategies: Random or cluster based with variant aligned orientations

Model-specific parameters: Couzin Model

After setting the general parameters, we now consider the model-specific parameters. As a short recap, Couzin et al. [3] formulated the motion of individuals in a group according to a zonal model, see figure 4. Within the zone of repulsion (zor), the objects maintain a minimum distance from each other and move away from each other to avoid collisions. Objects resemble their motion within the zone of orientation (zoo) and inclining each other in the zone of attraction (zoa).

These parameters are adjusted interactively using the sliders in the `Parameter Controls` see figure 4a),

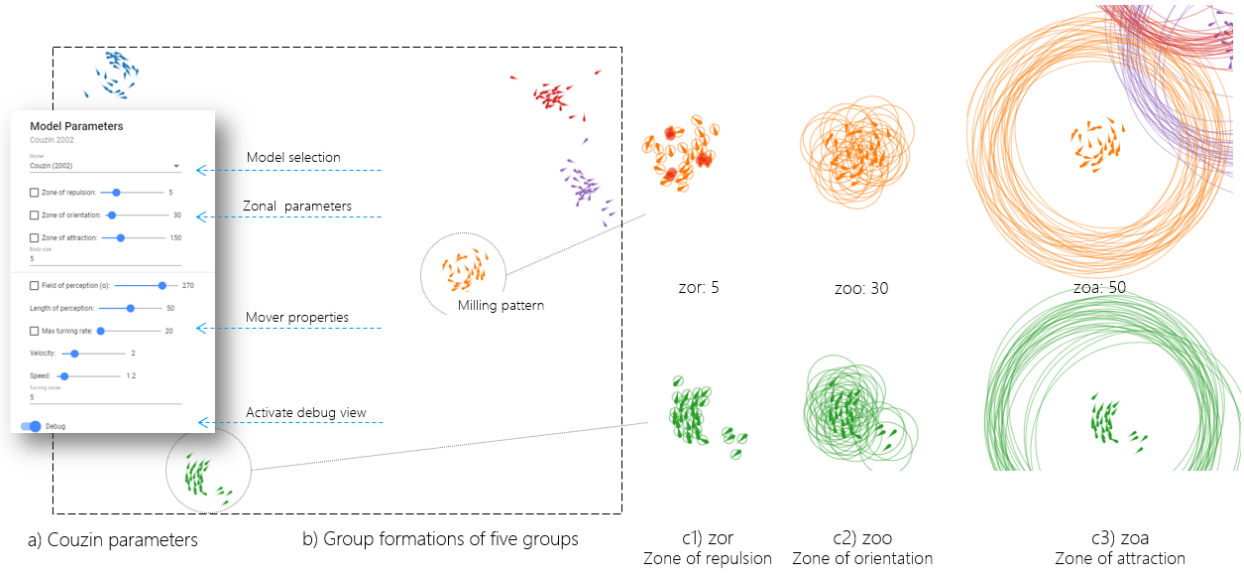


Figure 4: Couzin model: Group movement and zonal ramification

and the values correspond to the numerical unit system of the arena. The speed and the velocity can be interactively adapted, as described in the model, and the extent of noise can be defined to create natural casual wander behavior. The different zones, as well as the mover properties, see figure 4a), influence and generate the model-specific dynamics. To be able to visually reproduce these movements, a debug-toggle is available in the parameter controls. The individual parameters and their effects can be made visible using checkboxes. The individual zones are visually shown in the debug view, see Figure 4c1)-c3). If agents are

within the repulsion zones 4c1), a collision occurs - shown with a red filled circle. This figure shows that milling can occur (figure 4b), for example, within a relatively small zone of orientation (zoo in figure 4c2)), but large zone of attraction (zoa in figure 4c3)).

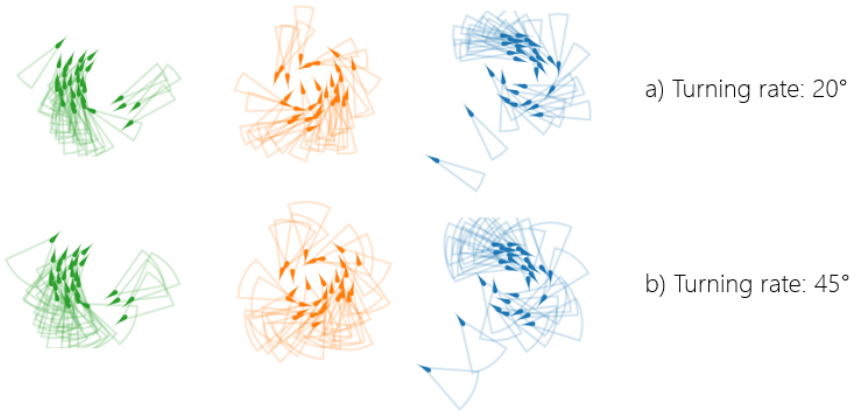


Figure 5: Couzin model: Variants of turning rate

But not only the individual zones of the Couzin model influence the generated behavior, but also mover parameters, such as the maximum possible turning rate and the field of view. Figure 5 shows the debug view of the turning rate. The smaller this parameter is set, the more restricted the mover is in its movement behavior and cannot break out as easily, but takes longer to rotate once around its axis. Figure 6 shows the sight field of movers. The larger the field of perception, the more the mover can perceive and the smaller is its blind spot. The blind spot corresponds to the non-gray area and represents the area a mover can't perceive. As soon as a mover sees another agent of its group, the vision field is colored in the cluster hue. If there is no interaction, the field of vision remains gray.

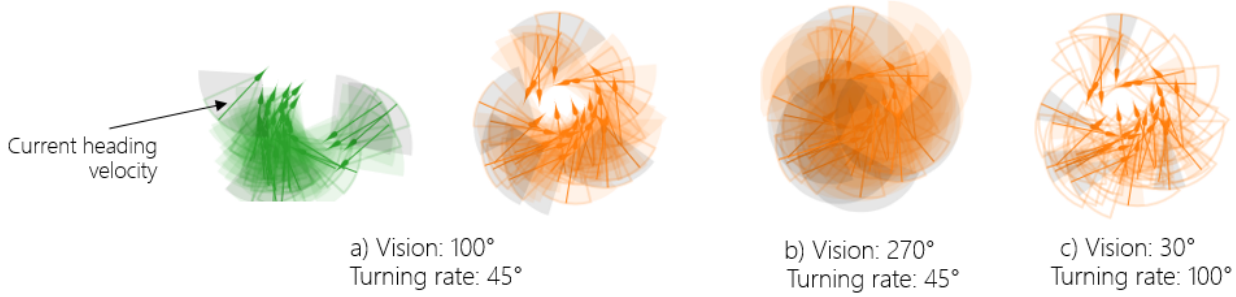


Figure 6: Couzin model: Combinations of the field of vision and turning rate. A small turning rate corresponds to small rotational changes and the larger the field of perception, the more the mover can perceive and the smaller its blind spot.

Model-specific parameters: Reynolds Model

Unlike the Couzin model, the Reynolds model is based on physical steering forces and the combination, as well as the interweaving, of different patterns. This model, however, forms a basis for the zonal model, which has been extended by agent characteristics such as the field of view and the restriction of the turning rate. Figure 7 shows on the left side the model parameters of the Reynolds model in the parameter controls. The basic parameters of the model are vector-based forces that are limited in their magnitude by the parameter `max speed`and which are added to the boids acceleration. Before applying the acceleration force to the velocity, its influence is anon limited by the parameter `max force`, which defines how much the force maximally affects the velocity caused by the acceleration.

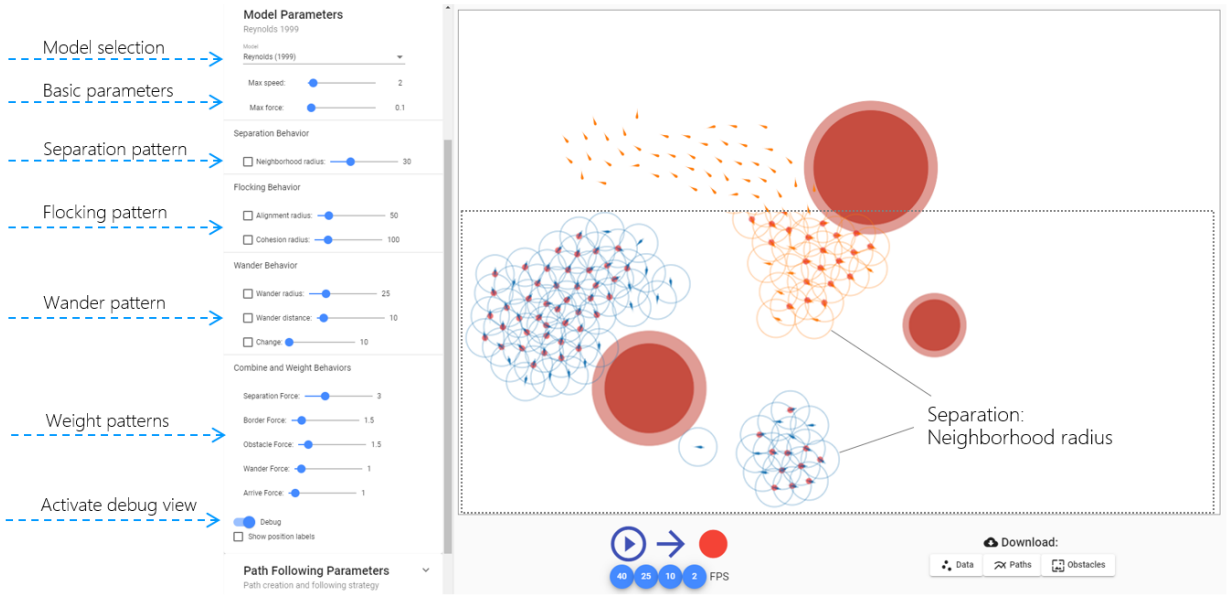


Figure 7: Reynolds model: Overview of model parameters and flocking pattern with collision visualization

The model parameters are divided into groups partitioned by patterns (separation, flocking and wander behavior), which is pointed out in figure 7 on the left, and which resulting forces can be weighted. The boid movement consists of various patterns, that each has its parameters, see table 3:

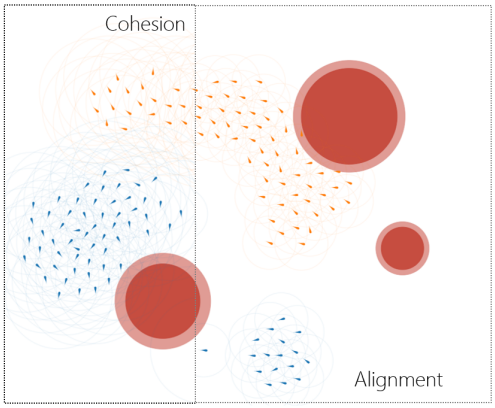
- **Separation:** boid changes its direction when it tends to collide with others in an adjacent neighboorhood. (Corresponding to zone or repulsion)
- **Alignment:** movers resemble their orientation with adjacent agents to move aligned as a group in a similar or the same direction (corresponds to the zone of orientation)
- **Cohesion:** a mover is attracted from others and changes its direction to approach them (corresponds to the zone of attraction)
- **Flocking:** the combination of separation, alignment, and cohesion to create collective movement behavior. Boids move in polarized groups in similar directions, align with others and are attracted by others.
- **Wander:** boid moves randomly around its environment. The pattern aims to produce realistic *casual* movement behavior by adding noise.

- **Seek:** mover seeks a target and the velocity changes towards it, as it is forced to the desired direction.
- **Path Following:** the mover intends to follow a path of motion, but if it is too far apart from the path, it seeks the trail and steers towards it.
- **Arrive:** a boid slows down when it approaches the desired target and stops as soon the desired location is reached.
- **Obstacle and Collision avoidance:** when the mover tends to collide with an obstacle or a wall, it steers away from it. The closer a boid is to the barrier, it slows down and starts turning to move away from it

Basic parameters: Analogous to the previous model, the application also offers the possibility to visualize individual parameters by activating the debug view, see figure 7. Collisions occur within the `neighborhood radius`, that corresponds to the zone of repulsion, and are also illustrated by red filled circles (figure 7 activated debug view in the arena). The direction change takes place in the pattern `separation`.

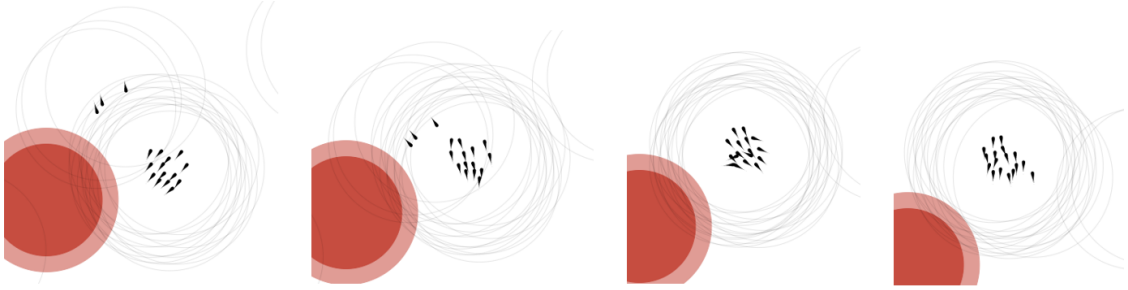
Seperation Pattern: When boids collide, they move away from the colliders and the size of the neighboorhood zone determines their density. The higher the separation zone, the less dense the group is.

Flocking pattern: To generate collective swarming behavior, Reynolds defined the `flocking pattern`, which is composed of `separation`, `alignment`, and `cohesion` and corresponds to the zones in the Couzin model. Figure 8 shows the visually represented radii: `alignment` and `cohesion radius`. These parameters can be interactively changed by sliding and different variants can be explored. A sequential time series in figure 9 shows how flocking is achieved over time. When adjacent objects are in the alignment radius of the mover, they align. Also, objects remain in their small groups, but with increased cohesion radius, they attract objects, as can be seen in figure 10: an agent affiliates with another group of movers.



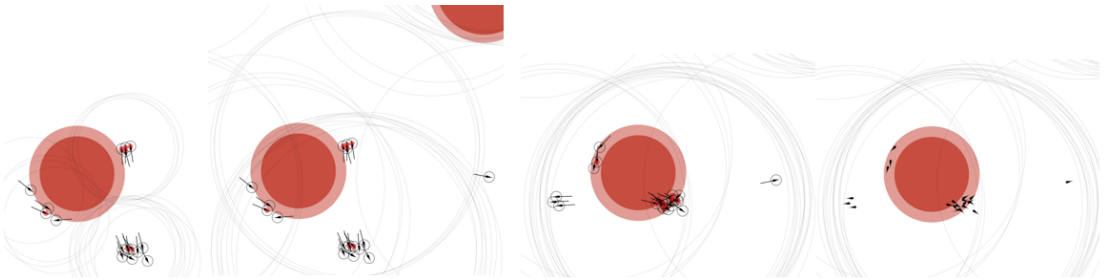
Flocking pattern:
Neighborhood: 30, Alignment: 50, Cohesion: 100

Figure 8: Reynolds model: Flocking pattern - visual representation of alignment and cohesion radius



Flocking pattern: Sequential steps of alignment
Neighborhood: 10, Alignment: 50, Cohesion: 100

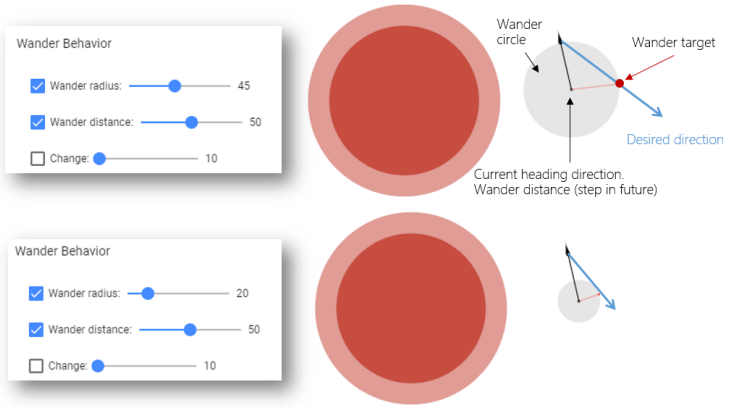
Figure 9: Reynolds model: Flocking pattern - sequential group formation. Distinct groups attract each other within the cohesion zones and align their velocity to move collectively.



Flocking Pattern: Increasing cohesion zone attracts mover

Figure 10: Reynolds model: Flocking pattern - Increasing group size by cohesion. Distinct groups move separately and by changing the size of the cohesion radius, these groups are attracted to each other and pull in a single mover.

Wander pattern: Reynolds created the `wander pattern`, to create natural and casual movement. On the one hand, it adds noise towards the motion to make it more realistically and by parametrization, on the other hand, it can represent the turning rate of a boid. As seen in figure 11, the pattern consists of three parameters. The `wander radius` defines the maximum turning rate. The bigger the diameter, the abler the mover is to turn. For comparison see the desired heading directions marked in blue: the rotation rate is higher in the upper example. To determine the desired direction, a point in the future with a `wander distance` based on the current velocity is taken and noise (`change` from 0°-360°) is added towards it, that results in the `wander target`. Starting from the current position of the boid heading towards the wander target defines the new desired direction and the velocity shifts there. By activating the debug option of the model, these parameters and the pattern can be visually explored.



Wander pattern: Emulates *casual* wander behavior.
Steers velocity towards desired direction (blue)

Figure 11: Reynolds model: Wander pattern. Emulates casual movement by adding noise to the velocity and determining the turning rate by the diameter of a wander radius. The higher the radius, the higher to rotational ability.

Obstacle and wall collision avoidance: In contrast to the zonal model, Reynold's approach already contains obstacle avoidance and path following. The application comes with different pre-defined obstacles, each having an offset, and the user can activate obstacle avoidance or turn this option off. When movers tend to cross the offset, they are forced to perform the `flee` pattern. In figure 12 different offset possibilities are represented by an interactive parameter that multiplies the obstacle offset by the chosen value. The larger the offset, the sooner the agents are steered away. But this can lead to unwanted behavior in combination with path following, as it pushes movers away from the desired path. The same technique was used to avoid wall collisions. When the boid approaches the border offset, that also can be parametrized, it steers away from it. This earlier turn-off results in a smoother flow of movement, as the agents are not abruptly repelled by the wall.

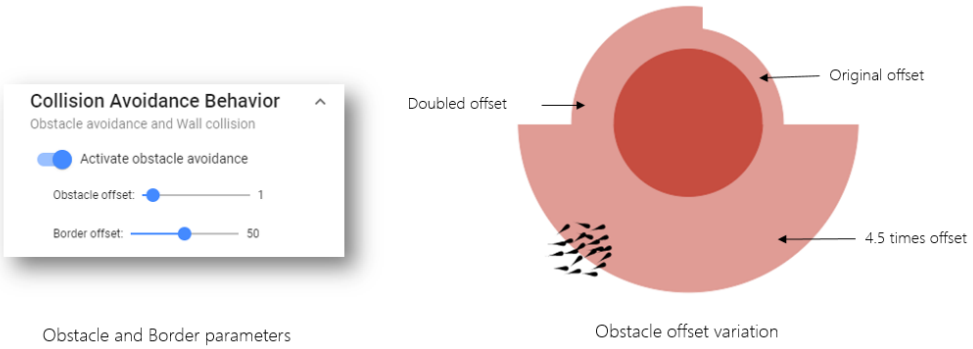


Figure 12: Reynolds model: Obstacle avoidance. Boid steers away when its located within the obstacle offset. This principle is also used to avoid wall collisions. The obstacle, as well as the border offset, can be interactively varied.

Path-following: To follow certain motion paths, the path following option first must be activated and the path-following control parameters are shown, see figure 13. The application comes with three differently shaped predefined paths to represent different motion patterns, the user can choose from and the ability to define the path following entrance. Either all boids move to the path start or they are heading to the starting point of the closest segment. A path has a `path radius`, that forces the agents to stay within the trail, as well as a `segment passing radius` to follow the path in the predefined sequential order. The path radius, as well as the radius of the segment transition point, can be controlled interactively. The narrower the path is, the more the objects are forced towards the centerline of the path. If the segment passing point is selected too small, the movers may have to return to this point to pass the segment.
Various path-following outcomes are shown in figure 14. Path following is visually supported by the checkbox `Show Path Following` under the debug-toggle in the model parameter controls. The boid seeks the normal point on the path segment and the magnitude of the normal is calculated. If the distance to the normal point is greater than the path radius, the boid performs the pattern `seeks` to the segment end.

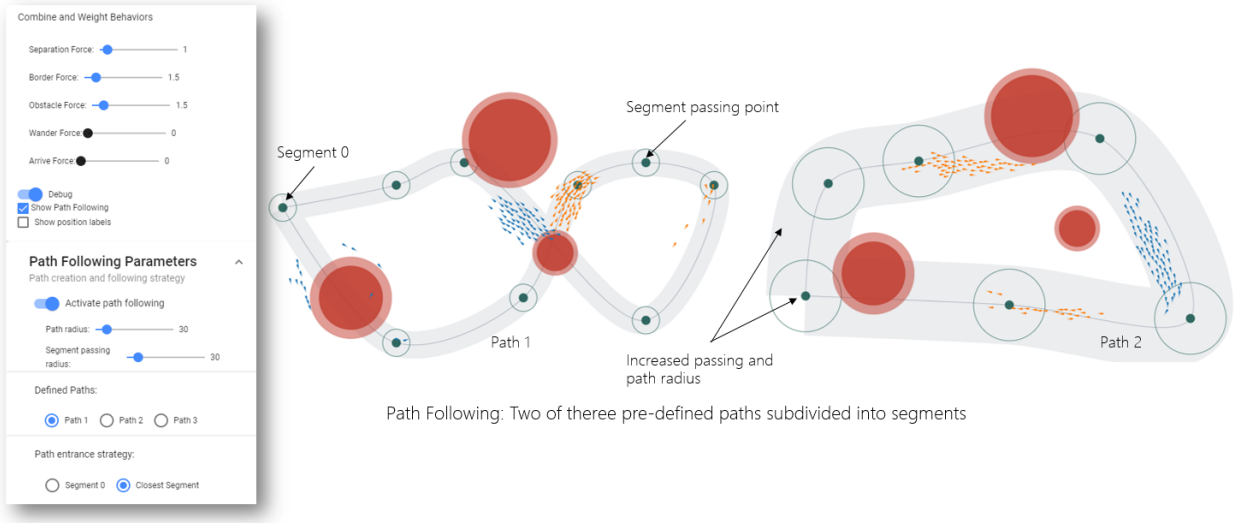


Figure 13: Reynolds model: Two of the three predefined paths and path following settings

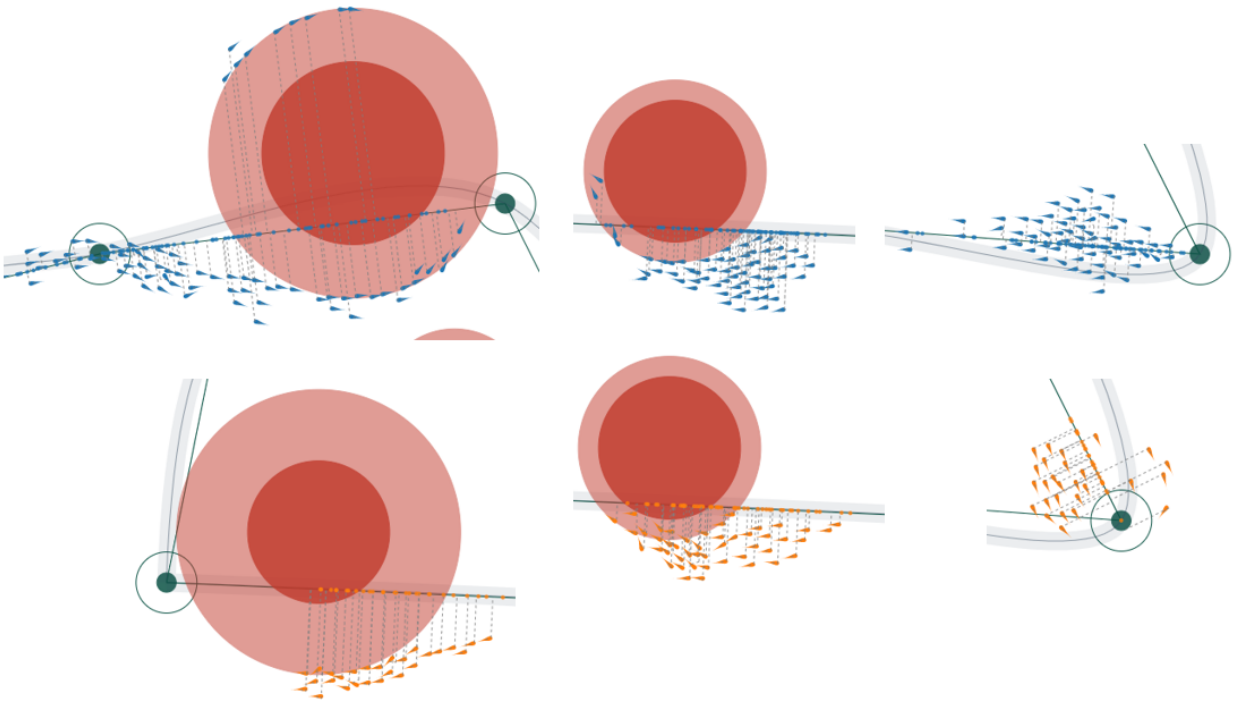


Figure 14: Visual examples of path following, as well as in combination with obstacle avoidance

Combine and weight behaviors: Each of the resulting forces, namely `separation`, `border`, `obstacle`, `wander` and `arrive force` (e.g. figure 12), can be weighted. Therefore, the forces are multiplied with is respective scalar values, ranging e.g. from 0 to 10. All forces are added to the boid's acceleration and the acceleration magnitude is limited towards the max force parameter before it is applied to the velocity. The separation force e.g. results from the collision avoidance pattern separation. To avoid an out-cancellation, the force should be greater than 1. Also, the border force, to avoid collisions with the wall should be greater than 1. To enable a smooth repulsion from a border, the arrive force slows down the speed of the boid, when approaching the border offset. So these three forces are combined used to avoid collisions and should not be set lower than 1. The obstacle avoidance force weighting depends on the offset variation. The bigger the offset, the lower the force can be set. If it is set too high, boids are forced to move exactly along the offset which can result in combination with other strong forces to a trembly semblance.

When path following is activated, the arrival force is set to 0, as no target is defined at which point the movers should slow down. Also, either path following or flocking is applied to the boids movement as cohesion behavior since the attraction comes from the path and not by others. To also create fluent behavior within the motion path following, the forces resulting from the wander and arrival pattern are not added towards the acceleration. The combination of the wandering behavior resulted in an unrealistic motion, not only within the path-following pattern, with trembly velocities, therefore it is implemented, but not taken into account in the project.

3 Parameters

The usage and outcome of the various parameters have already described in the usage section 2, but are summarized for a better overview and described in more detail in this section. Global parameters are listed in table 1. The arena size can be defined according to the desired dimensions and this coordinate system is used for the movement calculations and is stored in the exported dataset. These coordinates are scaled to the available screen dimensions to work without stipulation of minimum and maximum screen resolution.

General and global parameters	
Arena and screen size	
Arena width and height	Represents the *real* world. Calculations are done in arena coordinate system
Screen width and height	Scaled arena coordinates according to the available screen size
Mover	
No. Movers	Number of movers
Placement strategy	Choice of initial placement and group behavior
- Random	Random mover, randomly distributed in arena
- Cluster	Grouped movers. Number of movers equally distributed to cluster size
No. cluster	Number of clusters
Cluster orientation	Global orientation of cluster centroid in degrees (0°-360°)
Centroid variation	Individual mover orientation variation from the centroid orientation (0°-360°)
Collision Avoidance Behavior	
Obstacle offset	Reynolds: Offset boids are steered away from obstacle
Border offset	Offset to wall to simulate "soft" wall avoidance

Table 1: General and global parameters

The number of movers can be defined, as well as the placement method. The agents are either randomly distributed within the available arena space or can be segmented into a variable number of clusters. Based on the number of clusters, the agents are equally distributed in these groups, e.g. 300 movers in 3 clusters contain 100 movers each. Each cluster has a centroid and the agents are placed in different distributions around their corresponding cluster center. Also, each cluster has a global orientation and the individual mover's variation δ from this central orientation can be set. The variation is randomly generated ranging from 0 to the configured variation parameter and is added or subtracted from the centroid orientation θ:

$$\theta_{mover} = \theta_{cluster} \pm random(0, \delta)$$

A centroid orientation $\theta_{cluster}$ of 90° and a variation of $\delta = 10°$ results in initial movers' orientation ranging from 80° to 100°. The smaller the variation, the more aligned the movers start initially.

Other global parameters are collision avoidance offset. To avoid collisions with walls or obstacles, the mover must be able to detect impediments as soon as possible to adapt its velocity to steer away. The border offset is adrift of the arena dimensions to narrow them down. It *notifies* an individual about a possible collision to prevent a clash and an abrupt change in motion. Velocity change can be modeled by inverting the direction towards the wall in which the clash occurs. But this doesn't result in natural behavior and is cushioned through an earlier motion change that corresponds to the agent properties like the ability to turn.

3.1 Model-specific parameter space

To create a good dataset, natural behavior, as well as a realistic environment needs to be reflected. Following the mechanistic strategy according to Sumpter[9], two of the best known mechanistic models have been chosen as data generation foundation. Reynolds introduced behavior based on physical steering forces to adapt a boid's velocity[7]. To extend this model, Couzin et al.[3] created an extension that defines three hierarchical zones and defines a more biologically realistic method, as mover properties are covered in this

model. The main difference between these methods is, that behavior in the Couzin model is uniquely achieved by varying these parameters, whereas Reynolds defines patterns, representing intentions of boids, and forecasting targets. The individual parameter space of these models will be discussed in more detail than in section 2. To simulate movement, the application implements these two models and each has a move-method, that calculates and predicts the next spatial positions.

Model-specific parameters	
COUZIN MODEL	
Zonal properties	
Zone of repulsion (zor)	Collision area: Agent moves away from others that are in the zor
Zone of alignment (zoo)	Mover aligns with others that are within its alignment radius
Zone of attraction (zoa)	Movers within zone are attracted
Mover properties	
Field of perception	Agent's field of vision and blind spot (0°-360°)
Length of perception	Length of vision
Maximum turning rate	Limits the ability to turn (0°-360°)
Velocity	Distance mover can make in one calculation step or per tick
Speed	Speed parameter
Turning noise	Add noise to velocity to create *natural* behavior

Table 2: Couzin model parameters

Beginning with the Couzin model, it consists of very few parameters, listed in table 2, making the model minimalistic. Three behavioral zones, namely zone of repulsion (zor), zone of orientation (zoo) and zone of attraction (zoa) define the group movement of the agents. These zones define the motion in two hierarchical rules: Firstly, collisions with others are being prevented, secondly, the motion is aligned with neighboring movers. When movers enter the zone of repulsion, the mover avoids collisions by moving towards the opposite side of the central point of the colliders. This means, within the zor, the average position of all colliders for mover m is being calculated and mover m heads towards the opposite direction towards this mean position.

When no collision occurs, the mover resembles its velocity towards its adjacent neighbors within the remaining zones. The agent aligns with the average velocity of its neighboring agents within the zone of orientation and changes its desired direction towards the average spatial positions by adjacent neighbors within the zone of attraction. The resulting alignment and attraction velocities are averaged and define the desired velocity of mover m.

To take the agent properties into account, a mover only has a limited capability to turn within one time unit. This ability is expressed by the parameter `maximum turning rate`. The resulting velocities are translated into a global orientation, ranging from 0-360° and the final resulting velocity is limited by the turning rate. This means, when the desired velocity changes from 90° to 170°, but the mover's capability to turn is 10°, the desired velocity is set to 100°. To generate a more realistic behavior, noise is applied to the desired velocity, e.g. ±5°. Thus, the global orientation varies from 95° to 105° for mover m. As the model assumes constant speed and velocity, both parameters can be interactively set. The resulting direction vectors are normalized and the velocity expresses the distance in the desired direction and can be referred to as a movement step, which is regulated by the speed parameter.
To determine which movers are within the zone of alignment and attraction, the agent must be able to perceive its surrounding neighbors. This is expressed by the field of perception and can vary from 0° to 360°. Defining a field of vision of e.g. 270° contains a blind spot of 90° in which the agent can't perceive other individuals. To determine if mover m sees another mover d, its vector is defined and checked, if it lies within the perception range.

Model-specific parameters	
REYNOLDS MODEL	
General boid properties	
Max speed	Limits the speed per iteration
Max force	Limits the forces magnitude
Pattern: Separation	
Neighborhood / Separation radius	Collision avoidance zone: Steer away from colliders
Pattern: Flocking	
Alignment radius	Radius in which the movers' velocity is aligned with other
Cohesion radius	Attraction radius for cohesioning other agents
Pattern: Wander	
Wander radius	Extent of wander direction, corresponds to turning rate
Wander distance	Current velocity in x-units in the future
Change	Adds noise to the desired velocity
Pattern: Arrive	
Arrive distance	Distance towards target to slow down
Pattern: Seek (Flee: reverse)	
Target	Target that agent seeks. Velocity steered towards that target
Pattern: Path Following	
Path radius	Radius the movers shall adhere. Otherwise steered to center
Future position	Predicted position in the future
Normal point	Normal point on path segment
Path target	Future target point on path that agent seeks
Pattern: Obstacle and Collision avoidance	
Target	Target that mover tries to evade
Offset	Mover enters offset and "flees"

Table 3: Reynolds model parameters. Patterns define the behavior of boids and each pattern results in a force, that is combined with other applied patterns. This combined force is limited by boid properties, that limits the force to be applied to the boid's velocity.

In contrast to the zonal model, the Reynolds approach bases on physical steering forces. Behavioral movement is divided into basic patterns, each resulting in a force. The implemented patterns and parameters are shown in table 3. A boid has in each iteration step an acceleration of 0 and a starting velocity of 0, which is adapted in each step. Each pattern results in a physical steering force that is limited to maximum speed, which is added to the boid's acceleration. The speed represents the ability to move within one movement step.
After applying all patterns, the acceleration is limited to the parameter max force and is then added to the velocity. The force parameter regulates the effect of the force on the current velocity. The smaller the force, the less abrupt changes occur, but the longer it takes for changing the direction towards the desired velocity.
As in the zonal model, the highest priority is set to collision avoidance, which is expressed in the *separation* pattern. A neighboring radius is defined, which corresponds to the zone of repulsion, within this the boid steers away from colliding others. The closer the colliders approach, the bigger the magnitude of steering force, but which is also limited to the maximum force.
To create flocking and collective group movement behavior, the patterns separation, alignment, and cohesion are applied. The boids within an alignment radius and between this alignment zone and within a cohesion radius are determined and stored in two lists. For the alignment pattern, the velocities are averaged and the boid is steered towards this target. Similar to the alignment pattern, the locations of adjacent boids

are averaged in the cohesion pattern and the boid *seeks* this target location. The seeking pattern takes the current location and a target location and steers the boid limited by max force towards the desired target.

To simulate *realistic* and casual behavior, the model depicts the wander pattern that adds noise to the velocity and which parameters motivate the definition of the movers turning rate. The structure of the behavior pattern is illustrated in figure 11. A movement step in the future with the current velocity and a *wander distance* is predicted. At this future position, a circular area with a *wander radius* is defined on which the resulting wander target is quite randomly located based on a *change* parameter, that stipulates the *wander target*. The bigger the wander radius, the bigger the velocity variation. The circumference of the circle determines the maximum turning rate. As both patterns didn't result in the desired biologically realistic behavior, they are both implemented but do not influence the boidal movement.

To enable path following, each path is segmented and each boid knows its current path segment. To follow the path, the boid's future position is predicted and is forced to steer towards the normal point on the path if its distance from the predicted position to the normal point on the path's centerline is bigger than the path radius. Each segment has a starting and an endpoint, that the movers have to pass. The smaller the passing radius, the more likely that boids need to turn to pass the segment. As the steering forces are limited by the maximum force parameter, obstacles and walls have offsets, that cause the movers to depart earlier and prevent possible collisions.

Obstacles are avoided by *fleeing* from them. As soon as a boid enters the obstacle offset, it steers away from it. The same concept has been applied to avoid wall collisions. When the wall offset is hit, the mover's velocity changes. But not only its velocity: the closer the boid nears the wall, the slower it gets by performing the *arrival* pattern. When a target is defined, the boid has the ability to slow down when it approaches the target. Based on a distance to the desired target, the mover decelerates as the ratio of the distance d to the wall and the offset o determines the obstacle force magnitude Δ, which is normalized by the max speed parameter before to avoid an out-cancellation.

$$\Delta = \frac{d}{o}$$

These different from pattern resulting forces can be weighted. Since forces can cancel each other out, it is possible to weight them by priority. They are normalized and multiplied before adding to the acceleration. Initially in the prototype, the forces for separation and collision avoidance have a higher weight than the other patterns.

4 Setup

The application is uniquely front-end and doesn't require any backup, so the setup can be quite easily done[1]: a precompiled web project is available in the submission folder.

1. To run this application, navigate to the folder `VDG4CB` where the *index.html* must be contained.
   `cd home/dist/VDG4CB`
2. Start an webserver, e.g.
   `python -m http.server`
3. open the application in your browser at
   `http://localhost:8000/`

The project can be set up alternatively using `nodeJS` itself and the provided angular project in the `gitlab`[2]

[1] Please note, that this documentation refers to the standard ports. Once the ports have been changed, the commands need to be adapted.

[2] dbvis-gitlab: `https://gitlab.dbvis.de/piljek/master_project`

1. navigate to the angular project folder
   `cd VDG4CB`
2. start the application
   `ng serve --open`
3. that compiles and opens the prototype directly in your browser at
   `http://localhost:4200/`

When changes have been applied, the project can be built by `ng build`, which transforms the angular prototype with typescript to a web application using JavaScript in the `dist` folder, that can be executed as described. The application can be run on any web-server as well: Copy the dist folder and move it to the desired directory.

The application can be run on any web-server as well: Copy the dist folder and move it to the desired directory.

Development Setup

To extend the application, this section describes how to setup the development environment. The prototype is written in typescript, using Angular as web framework. A full list of the used technologies, as well as the used libraries and versions, are listed in table 15.

Technologies	Version	Description
Web framework and programming language		
Angular	~8.2.3	Web development platform
TypeScript	~3.5.1	Typed superset of JavaScript. Programming language with native OO-support, superset of ECMA Script 2015 (ES2015) syntax
Node.js	~12.3.1	Node.js: JavaScript runtime built on
npm	~6.9.0	Node package manager
RxJS	~6.4.0	library for reactive programming for an event-based approach
Visualizations		
Angular Material	~8.2.3	UI component and design components for Angular applications
hammer.js	~2.0.8	Animated gestures for Angular Material
Sass		Syntactically Awesome Style Sheets for styling html components
HTML	5.0	Hypertext Markup Language: description and definition of web content
d3	~5.9.2	JavaScript library for visualizing data using HTML, SVG,... Elements
Helper libraries		
ts-vector	~0.1.0	Typescript vector library
export-to-csv	~0.2.1	Export data to csv-file

Figure 15: Used technologies and libraries

To develop the prototype, the following was installed:

- node.js[3]. Check and verify installation, run the command `node -v`
- typescript[4] with the Node Package Manager (npm): `npm install -g typescript`
- Angular[5]: `npm install -g @angular/cli`
- d3[6]:

[3]node.js: `https://nodejs.org/en/` current version
[4]typescript: `https://www.typescriptlang.org/`
[5]Angular: `https://angular.io/`
[6]d3: `https://d3js.org/`

```
npm install -g --save d3
npm install -g --save-dev @types/d3
```

D3 needs further integration. Include the following to the file `package.json` (the content of this file is included at the appendix of this handbook, see section A):

```
"d3": "~5.9.2" in dependencies
"@types/d3": "~5.7.2" in devDependencies
```

Afterwards run `npm install` to build the project with d3-Integration.

- Angular Material[7]: `ng add @angular/material` and follow the setup.
  `HammerJS` is used for the sliding and toggle events. Afterwards the installed parts must be as well included in the `package.json` file:

```
"@angular/cdk": "~8.2.3"
"@angular/material": "^8.2.3"
"css-element-queries": "^1.2.1"
```

To further extend the application with components, services, or interfaces navigate to the desired installation directory and run in the commnand line

```
cd desired_folder
ng generate {component, interface, service,...} name
```

or shortly:

```
ng g {c, i, s} name
```

If a simple backend for development is needed, node provides a simple backend, that can be installed. Run `npm install -g json-server` in the command line. A JSON file must be placed into the root folder of the generated server, e.g. call it db.json. Start the installed server using `json-server --watch db.json`, it'll will defaultly run on port 3000, so a simple http-server to run the application as mentioned in section 4 can be started.

5 Code and Deployment

The application's code is documented using high-level class and method comments using TSLint[8] style. Since Angulars basic concepts are components, templates, directives, services and so on, the architecture defined in Milestone 2 has been adapted to this concept, see figure 16. There are three main components: `ArenaComponent`, as well as the `Parameter Controls` are situated . Each component has its HTML template with direct event and property binding. The most important components have been integrated with services that transport data and properties and enable communication between the components via observables using RxJS[9].

Furthermore, I implemented the visual objects like `Mover, Boids, ZonalMovers, Paths`, and `Obstacles` as components, because the object instances are bound to them by injection and the properties can be implemented directly using `d3` in the SVG elements within the templates. The resulting project structure is displayed in figure 17 and consists of the following structure:

- **interfaces**: Basic object classes and interfaces that are either used a export templates (e.g. obstacles) or that other classes inherit from (`mover, model`)

[7] Angular Material Design components: `https://material.angular.io/`
[8] TSLint: `https://palantir.github.io/tslint/`
[9] RxJS: https://rxjs-dev.firebaseapp.com/

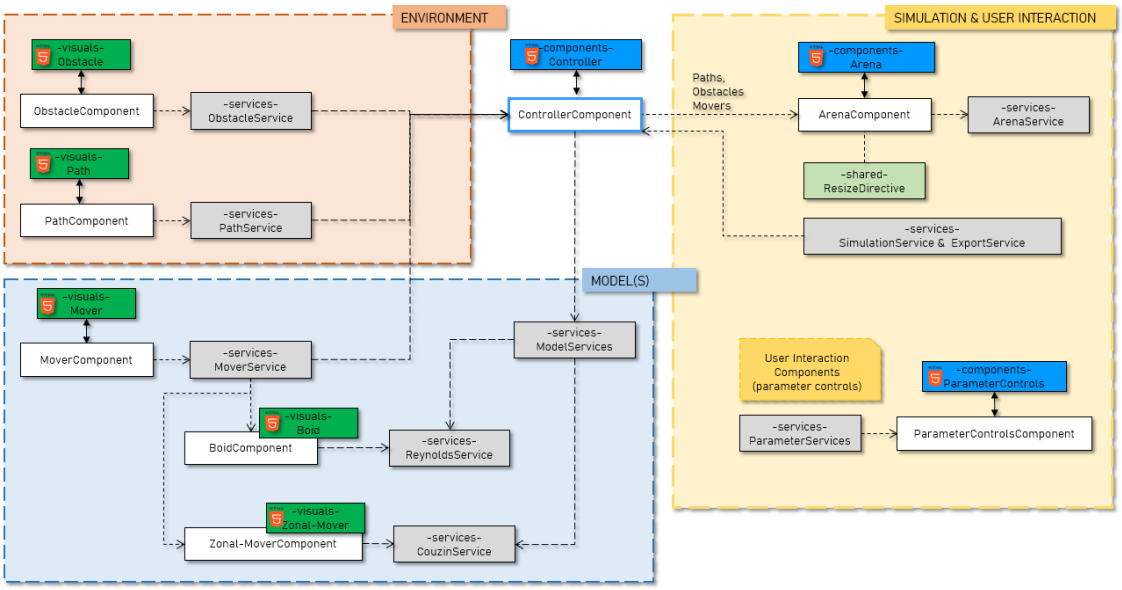


Figure 16: Implementation architecture

- **components**: Contains the controlling components *controller, arena* and navigational *parameter controls (path, obstacle, parameters*
- **services**: Services for the above mentioned components, as well as model related implementations
- **visuals**: Visual components and HTML templates for obstacles, paths and movers, as well as model specific properties
- **shared**: Helper classes as resize directives or linear algebraic calculations

```
components
|-- controller
|-- arena
|-- controls
    |-- control-obstacle
    |-- control-path
    |-- parameter-controls

shared
|-- lin-alg
|-- resized.directive
|-- resized-event

services
|-- arena
|-- export
|-- models
    |-- Reynolds1999
    |-- Couzin2002
|-- mover
    |-- boid
    |-- zonal
|-- obstacles
|-- parameter-controls
    |-- obstacle
    |-- path
    |-- parameter
|-- path
|-- position
|-- simulation

interfaces
|-- export
    |-- export
    |-- export-path
|-- model
|-- mover
|-- obstacle
|-- path
|-- segment

visuals
|-- boid
|-- mover
|-- obstacle
|-- path
|-- zonal-mover
```

Figure 17: Code structure

Input Parameters

The default and initial parameter settings, such as the arena size or model related properties are all stored in the `assets` folder. To access the folder inside the Angular project, it is integrated into the `compilerOptions` in the project file `tsconfig.json`:

```
"paths": {
      "@assets/*": [
        "src/assets/*"
      ]
    }
```

and can be imported in classes or interfaces.

Obstacles can be added or removed from the `assets/obstacles.json`. An obstacle has a unique identifier, a spatial position, a radial size and an offset that forces to movers to steer away The file structure looks like:

```
[
    {
        "id": 1,
        "x": 750,
        "y": 300,
        "r": 40,
        "offset": 10
    },...
]
```

The general initial parameter settings are defined in `assets/vdg-config.ts`. Further parameters can be added or existing ones can be changed. The following parameters are available in the configuration file:

```
const CONFIG = {
    N: 50,                  // number of movers
    PLACEMENT: 'RANDOM',    // placement method: RANDOM or CLUSTER
    CLUSTER: 3,             // number of cluster
    ARENA_WIDTH: 1200,      // real world arena width
    ARENA_HEIGHT: 800,      // arena height
    SCREEN_WIDTH: 600,      // screen dimensions for intialization
    SCREEN_HEIGHT: 300,     // screen dimensions for intialization
    WALL_OFFSET: 50         // offset to avoid wall collision,e.g.start
                            // avoidance maneuver
};

export default CONFIG;
```

Paths can be added or adapted in the `services/PathService`: A list of points is created and segmented. This procedure defines closed paths, as the endpoint of the last segment is the path start.

```
constructor(private navigationPathService: NavigationPathService) {
    this.paths = new Array();
    this.paths.push(this.createPath())
...
}
// Create list of points that are further segmented
public createPath() {
    let points = new Array();
    points.push(new Vector(150, 400));
```

```
    ...
    points.push(new Vector(400, 100));
    return new Path(pId, this.createSegments(points));
}

// Create path segments with start and endpoint. Generates a closed path.
public createSegments(points: Vector[]) {
    let segments = new Array();
    let index = 0;
    for (let i = 0; i < points.length; i++) {
      let p0 = points[i % points.length];
      let p1 = points[(i + 1) % points.length];
      let s = new Segment(index, p0, p1);
      segments.push(s);
      index++;
    }
    return segments;
}
```

Initial model parameters are specified as well in a configuration file in the assets folder: (`assets/vdg-models.ts`)

```
export const MODELS = {
 MODEL_COUZIN_2002: {
    id: 'Couzin (2002)',        // model id
    zor: 5,                     // zone of repulsion: collision avoidance behavior
    zoo: 30,                    // zone of alignment: coordinated polarized group movement
    zoa: 50,                    // zone of attraction: attraction behavior
    perception_field: 270,      // field of vision ranging from 0-360 degrees
    perception_length: 50,      // length of perception: code related to zoa
    velocity: 2,                // mover's velocity
    speed: 1.2,                 // agent speed
    turning_rate: 20,           // animal ability to turn in degrees (0-360 degrees)
    turning_noise: 5,           // adds noise to turning_rate (+- noise)
    body_size: 5                // spatial extent of mover
 },
 MODEL_REYNOLDS_1999: {
    id: 'Reynolds (1999)',      // model id
    mass: 20,                   // boid mass
    velocity: 3,                // agent velocity
    max_force: .1,              // limitation of steering force
    max_speed: 2,               // maximum speed of mover
    neighborhood_radius: 10,    // collision radius

    // wander behavior
    wanderRadius: 25,           // variation of desired direction
    wanderDist: 10,             // step size for predicted movement step in next iteration
    wanderChange: 10,           // noise to resulting desired direction

    // flocking
    alignment_radius: 50,       // alignment radius for coordinated movement in flocking
    cohesion_radius: 100,       // cohesion radius in which neighbours are attracted
```

```
    // forces
    separationForce: 1,         // weigthing separation force --> multiplication of force
    borderForce: 1.5,           // weighting force to flee from walls
    obstacleForce: 1.5,         // weighting obstacle avoidance force
    wanderForce: 0,             // disabling wander force due to unrealistic behavior
    arriveForce: 1              // weighting arrival force when boid approaches wall
 }
};
```

To integrate new models, the following has to be done:

1. Include parameters in `assets/vdg-models.ts`. It must have at least the field `id`
2. Define the parameter controls for the new model in a new component or in `component/control-parameter.component.html` and implement the paramater change methods in the respective typescript class
3. Create a service that wraps the parameters in an observable subject using RxJS
4. Model a mover class that extends the `mover` superclass and create a visual component to display mover settings using d3.
5. Adapt the `MoverService` to place your mover type
6. Create model that inherits the interface `model` and implement the `move()`-method

References

[1] Natalia Andrienko and Gennady Andrienko. Designing visual analytics methods for massive collections of movement data. Cartographica: The International Journal for Geographic Information and Geovisualization, 42(2):117–138, 2007.

[2] Juri Buchmüller, Dominik Jäckle, Eren Cakmak, Ulrik Brandes, and Daniel A Keim. Motionrugs: Visualizing collective trends in space and time. IEEE transactions on visualization and computer graphics, 25(1):76–86, 2018.

[3] Iain D Couzin, Jens Krause, Richard James, Graeme D Ruxton, and Nigel R Franks. Collective memory and spatial sorting in animal groups. Journal of theoretical biology, 218(1):1–11, 2002.

[4] Somayeh Dodge, Robert Weibel, and Anna-Katharina Lautenschütz. Towards a taxonomy of movement patterns. Information visualization, 7(3-4):240–252, 2008.

[5] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. Visual analytics: Definition, process, and challenges. In Information visualization, pages 154–175. Springer, 2008.

[6] Craig W Reynolds. Flocks, Herds and Schools: A Distributed Behavioral Model, volume 21. ACM, 1987.

[7] Craig W Reynolds. Steering behaviors for autonomous characters. In Game developers conference, volume 1999, pages 763–782. Citeseer, 1999.

[8] Manuel Stein, Halldor Janetzko, Andreas Lamprecht, Thorsten Breitkreutz, Philipp Zimmermann, Bastian Goldlücke, Tobias Schreck, Gennady Andrienko, Michael Grossniklaus, and Daniel A Keim. Bring it to the pitch: Combining video and movement data to enhance team sport analysis. IEEE transactions on visualization and computer graphics, 24(1):13–22, 2017.

[9] David JT Sumpter. Collective animal behavior. Princeton University Press, 2010.

[10] Marlee A Tucker, Katrin Böhning-Gaese, William F Fagan, John M Fryxell, Bram Van Moorter, Susan C Alberts, Abdullahi H Ali, Andrew M Allen, Nina Attias, Tal Avgar, et al. Moving in the anthropocene: Global reductions in terrestrial mammalian movements. Science, 359(6374):466–469, 2018.

[11] Isobel Watts, Máté Nagy, Robert I Holbrook, Dora Biro, and Theresa Burt de Perera. Validating two-dimensional leadership models on three-dimensionally structured fish schools. Royal Society open science, 4(1):160804, 2017.

A package.json

```
{
  "name": "vdg4-cb",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~8.0.0",
    "@angular/cdk": "~8.2.3",
    "@angular/common": "~8.0.0",
    "@angular/compiler": "~8.0.0",
    "@angular/core": "~8.0.0",
    "@angular/forms": "~8.0.0",
    "@angular/material": "^8.2.3",
    "@angular/platform-browser": "~8.0.0",
    "@angular/platform-browser-dynamic": "~8.0.0",
    "@angular/router": "~8.0.0",
    "css-element-queries": "^1.2.1",
    "d3": "~5.9.2",
    "export-to-csv": "^0.2.1",
    "hammerjs": "^2.0.8",
    "rxjs": "~6.4.0",
    "ts-vector": "^0.1.0",
    "tslib": "^1.9.0",
    "zone.js": "~0.9.1"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~0.800.0",
    "@angular/cli": "~8.0.2",
    "@angular/compiler-cli": "~8.0.0",
    "@angular/language-service": "~8.0.0",
    "@types/d3": "~5.7.2",
    "@types/jasmine": "~3.3.8",
    "@types/jasminewd2": "^2.0.8",
    "@types/node": "~8.9.4",
```

```
    "codelyzer": "^5.2.0",
    "jasmine-core": "~3.4.0",
    "jasmine-spec-reporter": "~4.2.1",
    "karma": "~4.1.0",
    "karma-chrome-launcher": "~2.2.0",
    "karma-coverage-istanbul-reporter": "~2.0.1",
    "karma-jasmine": "~2.0.1",
    "karma-jasmine-html-reporter": "^1.4.0",
    "protractor": "~5.4.0",
    "ts-node": "~7.0.0",
    "tslint": "^5.15.0",
    "typescript": "^3.4.5"
  }
}
```

B General Parameters

: assets/vdg-config.ts

```
const CONFIG = {
    N: 50,                  // number of movers
    PLACEMENT: 'RANDOM',    // placement method: RANDOM or CLUSTER
    CLUSTER: 3,             // number of cluster
    ARENA_WIDTH: 1200,      // real world arena width
    ARENA_HEIGHT: 800,      // arena height
    SCREEN_WIDTH: 600,      // screen dimensions for intialization
    SCREEN_HEIGHT: 300,     // screen dimensions for intialization
    WALL_OFFSET: 50         // offset to avoid wall collision,e.g.start
                            // avoidance maneuver
};

export default CONFIG;
```

C Model Parameters

: assets/vdg-models.ts

```
export const MODELS = {
 MODEL_COUZIN_2002: {
    id: 'Couzin (2002)',        // model id
    zor: 5,                     // zone of repulsion: collision avoidance behavior
    zoo: 30,                    // zone of alignment: coordinated polarized group movement
    zoa: 50,                    // zone of attraction: attraction behavior
    perception_field: 270,      // field of vision ranging from 0-360 degrees
    perception_length: 50,      // length of perception: code related to zoa
    velocity: 2,                // mover's velocity
    speed: 1.2,                 // agent speed
    turning_rate: 20,           // animal ability to turn in degrees (0-360 degrees)
    turning_noise: 5,           // adds noise to turning_rate (+- noise)
    body_size: 5                // spatial extent of mover
```

```
 },
 MODEL_REYNOLDS_1999: {
    id: 'Reynolds (1999)',      // model id
    mass: 20,                   // boid mass
    velocity: 3,                // agent velocity
    max_force: .1,              // limitation of steering force
    max_speed: 2,               // maximum speed of mover
    neighborhood_radius: 10,    // collision radius

    // wander behavior
    wanderRadius: 25,           // variation of desired direction
    wanderDist: 10,             // step size for predicted movement step in next iteration
    wanderChange: 10,           // noise to resulting desired direction

    // flocking
    alignment_radius: 50,       // alignment radius for coordinated movement in flocking
    cohesion_radius: 100,       // cohesion radius in which neighbours are attracted

    // forces
    separationForce: 1,         // weigthing separation force --> multiplication of force
    borderForce: 1.5,           // weighting force to flee from walls
    obstacleForce: 1.5,         // weighting obstacle avoidance force
    wanderForce: 0,             // disabling wander force due to unrealistic behavior
    arriveForce: 1              // weighting arrival force when boid approaches wall
 }
};
```