Prof. Dr.-Ing. Ralf Steinmetz
Multimedia communications Lab

Dipl. Inf. Robert Konrad
Polona Caserman, M.Sc.

TECHNISCHE UNIVERSITÄT DARMSTADT

Game Technology
Winter Semester 2017/2018

Exercise 13

General Information

- The exercises may be solved by teams of up to three people.
- The solutions have to be uploaded to the Git repositories assigned to the individual teams.
- **The submission date (for practical and theoretical tasks) is noted on top of each exercise sheet.**
- If you have questions about the exercises write a mail to game-technology@kom.tu-darmstadt.de or use the forum at https://www.fachschaft.informatik.tu-darmstadt.de/forum/viewforum.php?f=557

P13. Practical Tasks: AI (5 Points)

In this exercise, we program a simple world for artificial intelligences to work in. The first part of the AI task is building a part of the flocking algorithm to control boids in the world. The second part allows an AI to switch between different steering behaviors based on the current game state.

The code is provided for you, your task is to fill out the respective functions. The code can be found at https://github.com/TUDGameTechnology/Exercise13

Please remember to push into a branch called "exercise13".

You can find the solution code for the practical exercises at https://github.com/TUDGameTechnology/Solution13.git.

P13.1 Flocking – Cohesion/Seek and Separation/Flee (2 point)

In `Flocking.h/.cpp`, you can see that cohesion and separation for flocking can be implemented by seek and flee steering behaviors. You can find the (empty) implementations for seek and flee in `Steering.h`. Implement these functions.

Seek should accelerate at maximum acceleration towards the target position (= the output to the "linear" component of the steering output should be a vector pointing towards the target with magnitude `maxAcceleration`). Flee is the opposite of this behavior.

Note: Without these functions, the moon's behavior will do nothing and the boids will slow down and stop eventually.

P13.2 State Machine (3 points)

The moon object should have a state machine with the two following states: Wander and Follow. It starts in state Wander. If it is closer than 1 unit to the Earth object, it should switch to Follow.
In the state Wander, the moon AI should have a wander steering behavior. In the state Follow, it should have a follow steering behavior, with the Earth object as its target.

The state machine is setup by defining objects for states, actions, transitions and conditions. The state, action and transition objects are already set up for you. Flesh out the class MoonTransition to allow it to check the distance between the Earth and Moon objects. Add the correct transition object instances to the state machine so that it does the correct task. You can find the relevant source code in `Exercise.cpp`

T13 Theoretical Tasks: AI & Sound (5 Points)

T13.1 Steering behaviours (2 points)

You are given two objects, A and B, with the given positions and speeds. Fill out the following table of steering outputs if A is given B as its target. The steering output could be requested as a velocity or an acceleration. In both cases, it is a two-dimensional vector.

The maximal speed of A should be 2, the maximal acceleration of A should be 0.5.

Please Show Your Calculations.

Object	Location	Velocity
A	5, 6	3, 3
B	8, 2	3, 4

Steering behavior	Steering output (vec2)
Seek (kinematic)	
Flee (kinematic)	
Seek (dynamic)	
Flee (dynamic)	

Seek (kinematic)

Vector to target with length = maximal speed

$$\boldsymbol{Seek_{kin} = \frac{B - A}{|B - A|} \cdot v_{max}}$$
$$\boldsymbol{= \frac{1}{5} \cdot \begin{pmatrix} 3 \\ -4 \end{pmatrix} \cdot 2}$$
$$\boldsymbol{= \frac{2}{5} \cdot \begin{pmatrix} 3 \\ -4 \end{pmatrix}}$$

Flee (kinematic)

Negative of Seek

$$\boldsymbol{Flee_{kin} = -Seek_{kin}}$$
$$\boldsymbol{= \frac{2}{5} \cdot \begin{pmatrix} -3 \\ 4 \end{pmatrix}}$$

Seek (dynamic)

Vector to target with length = maximal acceleration

$$\boldsymbol{Seek_{dyn} = \frac{B - A}{|B - A|} \cdot a_{max}}$$
$$\boldsymbol{= \frac{1}{5} \cdot \begin{pmatrix} 3 \\ -4 \end{pmatrix} \cdot 0.5}$$
$$\boldsymbol{= \frac{1}{10} \cdot \begin{pmatrix} 3 \\ -4 \end{pmatrix}}$$

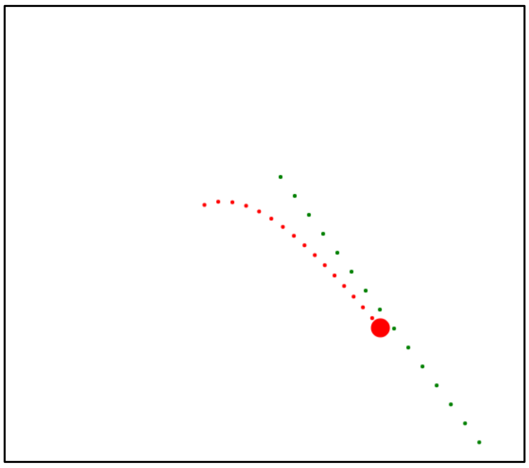
Flee (dynamic)

Negative of Seek

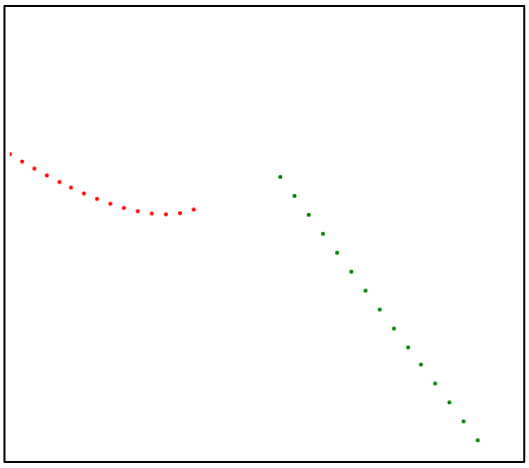
$$\boldsymbol{Flee_{dyn} = -Seek_{dyn}}$$
$$\boldsymbol{= \frac{1}{10} \cdot \begin{pmatrix} -3 \\ 4 \end{pmatrix}}$$

Examples (Red is A, green is B)
Try it out for yourself at [***https://wiki.ktxsoftware.com/wiki/exercises/gametech-ex13_steering.html***](https://wiki.ktxsoftware.com/wiki/exercises/gametech-ex13_steering.html)

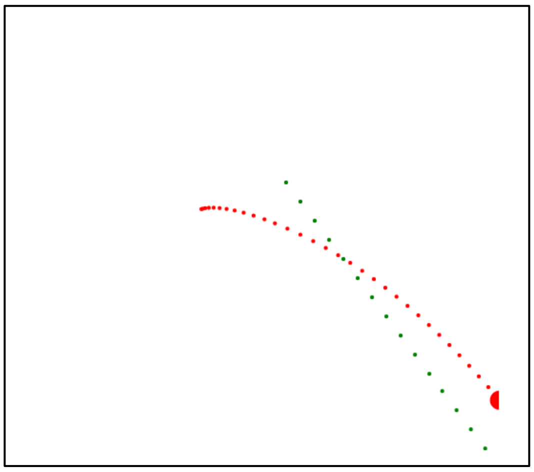
Kinematic Seek:



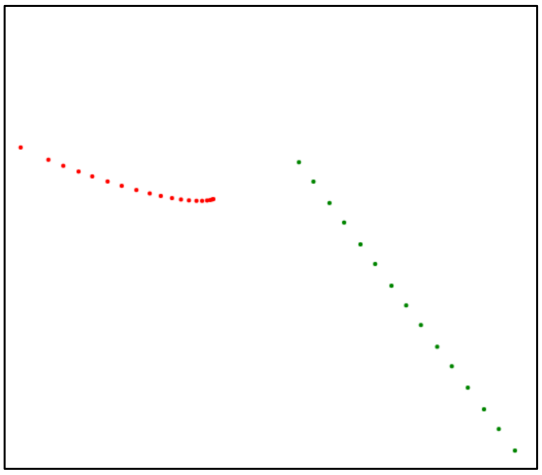
Kinematic Flee:



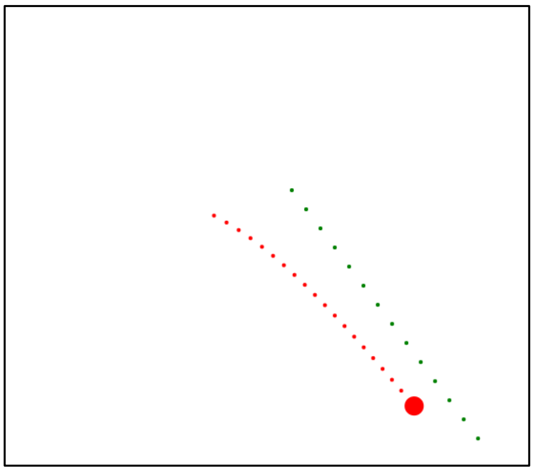
Dymamic Seek:



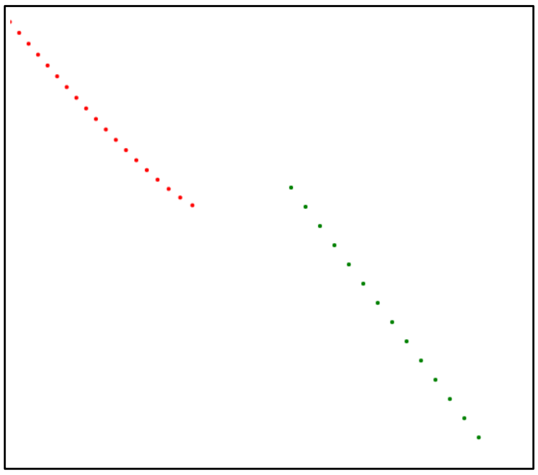
Dynamic Flee:



Pursue:



Evade:



T13.2 Pursue and evade (1 point)

Explain in your own words, the difference between pursue/evade and seek/flee.

For pursue and evade, we take our own max speed into account and compute how long we would take to reach the target if we travelled at full speed. Then, we compute the hypothetical position of the target if it moved forward with its current speed and use this new position as a target for a seek (pursue) or flee (evade) – use the dynamic variant of the seek and flee.

Hence, pursue and evade are similar to seek and flee, but also keep track of the current speed of the target and try to predict where it is moving to. Therefore, the seeking AI will get closer to the target but then fall behind. However, the pursuing AI will get closer to the position of the target and will eventually catch it.

T13.3 Movement algorithm identification (2 points)

Consider the provided pseudo-code for a movement behavior:

```
vec2 GetSteering(character c) {
      // Get orientation in degrees
      float orientation = c.orientation
      float randomDelta = randomInRange(-20, 20)
      float newOrientation = orientation + randomDelta
      return vectorFromAngle(newOrientation) * maxSpeed
}
```

a) Is this behavior kinematic or dynamic? Explain your answer. (1 point)

It is kinematic. We are using the maximal speed of the AI as the output, instead of providing an acceleration.

b) Which of the behaviors presented in the lecture is realized in the pseudo-code? Explain your answer. (1 point)

This is the Wander behaviour. We choose a new orientation that is slightly different from the previous orientation and move forward in this direction until the next time GetSteering is used. Also, no target position is used. Therefore, this pseudocode implements the Wander behaviour.