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

Dr. Florian Mehm
Dipl. Inf. Robert Konrad

TECHNISCHE UNIVERSITÄT DARMSTADT

„Game Technology“
Winter Semester 2014/2015

Exercise 14

For bonus points upload your solutions until **Friday the 13th of February 2015, 11:40**

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

1. Practical Tasks: AI (5 Points)

In this exercise, we program a simple world for AIs to work in. The first part of the 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 behaviours based on the current game state.

1.1 Flocking – Cohesion/Seek and Separation/Flee

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.

1.2 Flocking

In the code for the flocking steering behaviors, you can find the two parameters neighbourhoodSize and neighbourhoodMinDP. Examine the function Flock::prepareNeighourhood and explain what the purpose of the two parameters is in your own words.

1.3 State Machine

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

2. Theoretical Tasks: AI (5 Points)

2.1 Steering behaviours

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 velocity or 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.

Object	Location	Velocity
A	5, 6	1, 1
B	8, 2	3, 4

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

Note: 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.

2.2 Interruptible AI algorithms, Anytime algorithms

a) Can A* be interrupted during its execution and continued at a later time? If yes, describe why. If not, give a counterexample.

b) Imagine that an interruptible pathfinding algorithm is interrupted and continues 1 second later. Are there problems to be expected in this case? If so, what could the problems be?

c) If a) is true, does this make A* an Anytime algorithm? (i.e. Would the implementation be able to provide a valid path after being interrupted?)

2.3 AI LOD

Imagine a scene with many characters walking in random directions. In this scene, an AI LOD system is used. Nearby characters use collision avoidance, far away characters may freely interpenetrate.

What problems could arise if characters exploded when they collided?