CIS 462/CIS562 Computer Animation (Fall 2019)
Homework Assignment 8

(Behavioral Animation)

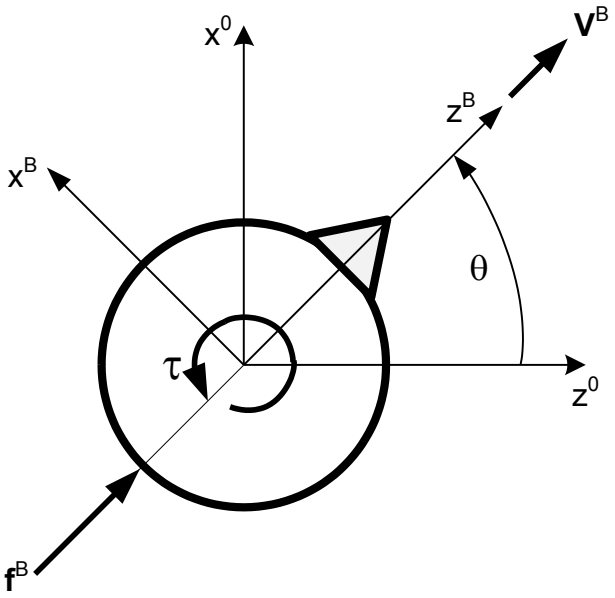
Due: Monday, Dec. 9, 2019 (must be submitted before midnight)

Please note the following:

- No late submissions or extensions for this homework.
- This is a programming assignment. You will need to use the *AnimationToolkit* code framework from the previous CIS462/562 assignments
- Double click on the file "Demo-BehaviorViewer.exe" in the AnimationToolkit\bin directory to see a demo of some of the basic behavioral animation system functionality you will need to implement in this assignment. Use the corresponding BehaviorViewer project to debug the functions you will need to implement in the ABehavior.cpp and BehaviorController classes.
- When you are finished with this assignment, commit and push the updated project files to your CIS462-562 GitHub repository.
- Work within the AnimationToolkit code framework provided. Feel free to enhance the AntTweakBar GUI interface if you desire to add more buttons or system variables
- You only need to implement the functions in the *.cpp files marked with "TODO" to complete this assignment.
- You only need to implement the functions in the *.cpp files marked with "TODO" to complete this assignment.
- **NOTE: THIS IS AN INDIVIDUAL, NOT A GROUP ASSIGNMENT. That means all code written by you for this assignment should be original! Although you are permitted to consult with each other while working on this assignment, code that is substantially the same as that submitted by another student will be considered cheating.**

ANIMATION TOOLKIT – BEHAVIORAL ANIMATION

The goal of this assignment is for you to become familiar with the implementation of behaviors for characters used in games and crowd simulations. In order to simplify the implementation, the root of the character (i.e. agent) is associated with a 2D disk-shaped vehicle called a "guide" as shown in the figure below. Each agent guide has mass (m) and moment-of-inertia (I) and is capable of generating a force $\mathbf{f}^B$ only along the body z-axis (z^B) and a torque τ only about the body y-axis (y^B). As a result of the applied force and torque, the agent velocity vector ($\mathbf{V}^B$) can be controlled with respect to the body z-axis (z^B) and its orientation (θ) can be controlled with respect to the world z-axis (y^B).



AGENT DYNAMICS

Equations of Motion (full 3D case) – See Aircraft Dynamics handout for more details

Translational Dynamics: $m(\boldsymbol{\omega}^B \times \mathbf{V}^B +$ $\dot{}$

Rotational Dynamics: $\mathbf{I}^B$ $\dot{}$ $\mathbf{I}^B\boldsymbol{\omega}^B = \boldsymbol{\tau}^B$

where m is the mass of the agent,

$\mathbf{f}^B$, $\mathbf{V}^B$ and $\dot{}$ are the force, velocity and acceleration represented in body axes,

$\boldsymbol{\tau}^B$, $\boldsymbol{\omega}^B$, $\dot{}$ are the torque, angular velocity and angular acceleration represented in body axes, and

$\mathbf{I}^B$ is the moment of inertia of the agent

Equations of Motion (2D planar Case) – what you need to implement in this assignment

Translational Dynamics: $m\dot{}$ where $\mathbf{f}^B = \begin{bmatrix} 0 \\ 0 \\ \mathbf{f}_z^B \end{bmatrix}$ and $\mathbf{V}^B = \begin{bmatrix} 0 \\ 0 \\ \mathbf{V}_z^B \end{bmatrix}$

Rotational Dynamics: $\mathbf{I}^B\dot{}$ where $\boldsymbol{\tau}^B = \begin{bmatrix} 0 \\ \boldsymbol{\tau}_y^B \\ 0 \end{bmatrix}$ and $\boldsymbol{\omega}^B = \begin{bmatrix} 0 \\ \boldsymbol{\omega}_y^B \\ 0 \end{bmatrix}$

and $\mathbf{I}^B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & 0 \end{bmatrix}$

Therefore, for the planar case of this assignment, this simplifies to:

Translational Dynamics: m

Rotational Dynamics: I_{yy}

Since the Animation Toolkit code is written to accommodate system dynamics of the form) for the full 3D case,

the state vector $\mathbf{x} = \begin{bmatrix} \mathbf{p}^0 \\ \boldsymbol{\theta} \\ \mathbf{V}^B \\ \boldsymbol{\omega}^B \end{bmatrix}$ and the control vector $\mathbf{u} = \begin{bmatrix} \mathbf{f}^B \\ \boldsymbol{\tau}^B \end{bmatrix}$,

where

$\mathbf{p}^0 = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is a vector representing the position of the agent in the world,

$\boldsymbol{\theta} = \begin{bmatrix} \theta_x & \theta_y & \theta_z \end{bmatrix}^T$ is a vector of Euler angles representing the orientation of the agent with respect to the world coordinate system,

$\mathbf{V}^B$ is the velocity or the agent in body coordinates, and

$\boldsymbol{\omega}^B$ is angular velocity of the agent with respect to world coordinates.

ASSIGNMENT DETAILS (140 points)

1. ***GUI:*** (10 points) In order to help you debug the behaviors as well as fine tune their responses, add relevant behavior parameters to the GUI for tweaking. Look at `BehaviorViewer::initializeGui()` function and add similar `TwAddVarRW` calls (along with their assocaiated callback functions) to connect the static (e.g. global) parameters for Behavior that you want to modify through the GUI.

2. ***Agent Body Dynamics***. (*10 points*) Implement the agent body dynamics in the `BehaviorController::ComputeDynamics` function to compute $\dot{\mathbf{x}}(t)$ and update the state to produce $\mathbf{x}(t + \Delta t)$ using Euler integration in the `BehaviorController::updateState()` function (this is similar to what you did in the particle system assignment). Also make sure you compute values for the world velocity $\mathbf{V}^0$ from $\mathbf{V}_z^B$ and $\boldsymbol{\theta}_y$ in the `ComputeDynamics` function so that position of the agent in the world ($\mathbf{p}^0$) can be updated in the `updateState` function.

3. ***Control Laws:*** (*20 points*) It is desired to control agent speed and direction (i.e. values of $\mathbf{V}_z^B$ and $\boldsymbol{\theta}_y$) using the command variables V_d and θ_d, respectively

   Given `Vdesired`, in the function `BehaviorController::control()` construct feedback controllers for $\mathbf{f}_z^B$ and $\boldsymbol{\tau}_y$ of the form:

$$u_1 = \mathbf{f}_z^B = m(K_v V_d - K_v \mathbf{V}_z^B)$$

$$u_2 = \boldsymbol{\tau}_y^B = I_{yy}(K_p \theta_d - K_v \dot{\theta}$$

   where V_d and θ_d are the magnitude and direction of `Vdesired`

such that:

a) in the steady state $\mathbf{V}_z^B = V_d$ and $\boldsymbol{\theta}_y = \theta_d$

b) the settling time for the V transient response is 0.4 sec, and

c) the settling time for the θ transient response is 0.25 sec, and there is no overshoot or oscillation.

Choose the appropriate gains for the velocity (V) feedback controller (i.e. K_v) and heading angle (θ) controller (i.e. K_p and K_v) that achieve the desired dynamic response described above. Make sure the values for the forces and torques generated are in the valid value range.

4. ***Behaviors*** (*100 pts*)

In this part of the assignment you will need to implement 6 types of individual behaviors and 5 types of group behaviors. Each behavior should utilize the environment information obtained during the Sense phase as part of the `BehaviorController::sense()` function (i.e. target location and location of obstacles) to compute values for `Vdesired` used in the Control phase.

a) (*50 pts*) Compute the desired velocity vector ($\mathbf{V}_{desired}$) by implementing the function `calcDesiredVel` for each of the behaviors listed below (which is then converted to the V_d and θ_d commands for the feedback controllers in the function `BehaviorController::control`):

1) (*5 pts*) Seek: $\mathbf{V}_{desired} = \mathbf{V}_{seek}$

2) (*5 pts*) Flee: $\mathbf{V}_{desired} = \mathbf{V}_{flee}$

3) (*10 pts*) Arrival: $\mathbf{V}_{desired} = \mathbf{V}_{arrival}$

4) (*10 pts*) Departure: $\mathbf{V}_{desired} = \mathbf{V}_{departure}$

5) (*10 pts*) Wander: $\mathbf{V}_{desired} = \mathbf{V}_{wander}$

6) (*10 pts*) Obstacle avoidance: $\mathbf{V}_{desired} = \mathbf{V}_{arrival} + \mathbf{V}_{avoid}$

See how the translational and rotational agent responses change when the values of the feedback controller gains K_p and K_v are changed. As a test, determine the value of K_p which makes the system go unstable.

b) (*30 pts*) Implement the `calcDesiredVel` functions for the following group behaviors:

1) (*10 pts*) Separation: $\mathbf{V}_{desired} = \mathbf{V}_{separation}$

2) (*10 pts*) Cohesion: $\mathbf{V}_{desired} = \mathbf{V}_{cohesion}$

3) (*10 pts*) Alignment: $\mathbf{V}_{desired} = \mathbf{V}_{align}$

c) (*20* pts) Once you have the separation, cohesion and alignment behaviors working properly use the desired velocity vector returned by these behaviors to implement the `calcDesiredVel` functions for following group behaviors:

1) (*10 pts*) Flocking

$$\mathbf{V}_{desired} = \mathbf{V}_{flock} = c_{sep}\mathbf{V}_{separate} + c_{align}\mathbf{V}_{align} + c_{coh}\mathbf{V}_{cohesion}$$

where $\mathbf{V}_{\text{separate}}$, $\mathbf{V}_{\text{cohesion}}$ and $\mathbf{V}_{\text{align}}$, are computed from behaviors in part b) above. c_{sep}, c_{coh} and c_{align} are constant coefficients. You should set them to appropriate values to achieve desired the behavior results. See what happens as you increase the number of vehicles in the flock, change the K_{cohesion} and K_{separate} gains of the cohesion and separate behaviors or change the gains of the feedback controllers (for example, reduce the heading K_p gain to make the agents less responsive and the K_v gain to make them more oscillatory).

2) (*10 pts*) Leader following

$$\mathbf{V}_{desired} = \mathbf{V}_{leader} = c_{sep}\mathbf{V}_{separate} + c_{arrival}\mathbf{V}_{arrival}$$

where $\mathbf{V}_{\text{separate}}$ and $\mathbf{V}_{\text{arrival}}$ are computed from behaviors in part a) and b) above. c_{sep} and c_{arrival} are constant coefficients that you will need to adjust to achieve the desired behavior result. Test the leader behavior by moving the target position for the leader in the world. See what happens as you increase the number of agents in the formation, change the K_{separate} gains or change the gains of the feedback controllers.

Please consult the class lecture notes and handout on Behavioral Animation for the implementation details of each of the behaviors listed above.

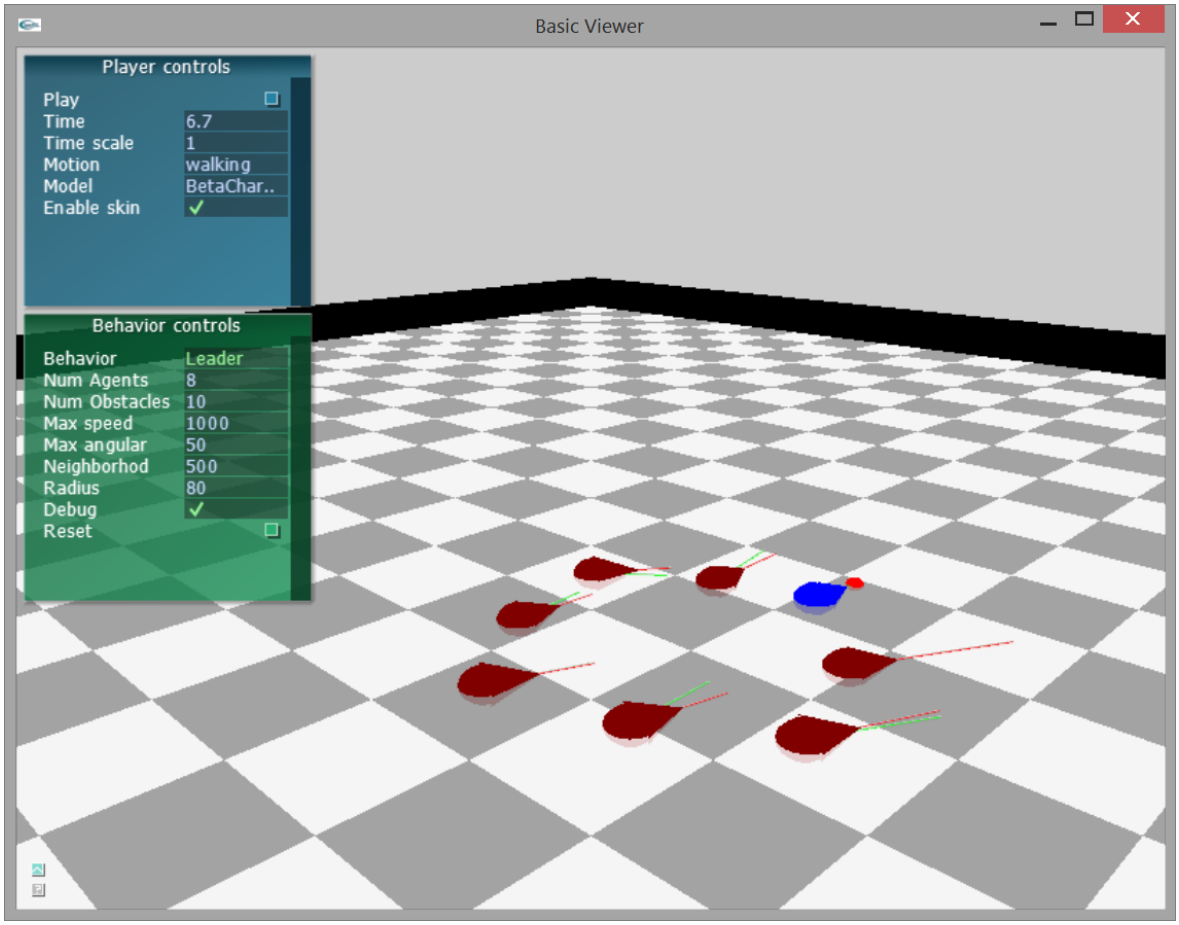
5. (Extra Credit: 10 points each) Try to make your own behaviors, e.g. formations, hide and seek, etc. Be creative!

About the BehaviorViewer Project

This assignment builds on the previous assignments, adding a new library and project applications. If you did not complete a previous assignment, you may edit the basecode project files to use the corresponding solution library. To do this, right click on the project, select Properties, and then expand the Linker options. Select 'Input' and then edit the 'Additional Libraries' to point to libAssignmentX-soln.lib instead of libAssignmentX.lib. You will need to do this for both the release and debug versions of your project.

User Interface Overview

The BehaviorViewer basecode includes a simple interface and 3D viewer which loads an agent models and associated animations as shown in the screenshot below. From this interface, you can change the number of agents, change their current behavior and modify various behavior parameters and gains. You will need to extend this interface with additional variables so you can easily tweak the behavior parameters and gains.



As before, the camera can be controlled with the mouse as follows:

- Left-button drag with the mouse to rotate
- Right-button drag with the mouse to pan
- Middle-button drag with the mouse to zoom
- 'f' button will focus the view

The red dot is the target used for each behavior. Ctrl-left click to move the target location. To

help with debugging your code, a check box for Debug View is provided to display a cone instead of the whole body for each agent. The red line indicates current velocity vector and the green line indicates the current desired velocity. You can also press the Pause button to stop the simulation or change the time scale parameter to value less than one to slow everything down.

Behavior Implementation Overview:

Agent movement is managed by the top level class BehaviorController which calls the active behavior to generate a desired velocity ($\mathbf{V}_{desired}$). This is used in turn to produced desired forces and torques in the Control phase which are then used to update the system state in the Act phase.