Dept. of Computer Science & Software Eng., Concordia University
COMP 476/6331 --- Winter 2020

Advanced Game Development

Assignment 1 --- Due Monday, January 27, 2020

PURPOSE: This assignment will give you the opportunity to learn more about programming AI behaviour. This assignment contains both **theoretical** questions, which you don't need to implement (doesn't mean you have to do them by hand), and a **practical** question, which requires you to write a Unity program.

SUBMISSION: The assignments must be done individually. On-line submission is required (see details at end) – a hard copy will not be read or evaluated.

PREPARATION: Parts of this assignment require knowledge of basic concepts from parts of the material covered in the course slides (exact sections of which books the material is based on are on the course schedule).

Question #1: (19%) [Theoretical Question]

For the following, assume that the center of an AI character is $\boldsymbol{p}_c = (5,6)$ and that it is moving with velocity $\boldsymbol{v}_c = (3,1)$. Also, assume that the stationary target is at location $\boldsymbol{p}_t = (8,2)$. Other parameters are the time between updates, $t = 0.4$, the maximum velocity, $v_m = 5$, and the maximum acceleration, $a_m = 17$. For a), b), d), and e), give some details as to how you compute the first position (detailed calculations are not needed for the rest of the positions).

a) (3%) What are the positions of the AI character at the next five updates if we use (Kinematic) *Seek* to reach the target? (Hint: No trigonometry is required for this and other questions like it, just simple vector arithmetic.)

b) (4%) What are the positions of the AI character at the next five updates if we use (Steering) *Seek* to reach the target?

c) (2%) For both (Kinematic) *Seek* and (Steering) *Seek*, describe the path followed by the AI character when trying to get to the target position (i.e., plot the paths and comment on the difference).

d) (4%) What are the positions of the AI character at the next five updates if we use (Kinematic) *Arrive* to reach the target? Use the blend of approaches I and II as described on Slide 66 of Week 1. Use the satisfaction radius $r'_{sat} = 1$ and time-to-target $t2t = 0.55$.

e) (5%) What are the positions of the AI character at the next five updates if we use (Steering) *Arrive* to reach the target? Use the arrival radius $r_a = 0.2$, slowdown radius $r_s = 1.5$, and a (different) time-to-target $t2t = 0.5$.

f) (1%) For both (Kinematic) *Arrive* and (Steering) *Arrive*, describe the path followed by the AI character when trying to get to the target position (i.e., plot the paths and comment on the difference).

Question #2: (6%) [Theoretical Question]

Suppose we have three characters in a V-formation with coordinates and velocities given by the following table:

Character	Assigned Slot Position	Actual Position	Actual Velocity
1	(22, 18)	(21, 16)	(3, 0)
2	(6, 13)	(5, 11)	(3, 2)
3	(29, 12)	(28, 9)	(6, 4)

a) (4%) First calculate the center of mass of the formation $\boldsymbol{p}_c$ and the average velocity $\boldsymbol{v}_c$. Use these values and the following Equation (with $k_{\text{offset}} = 1$) to calculate $\boldsymbol{p}_{\text{anchor}}$.

$$\boldsymbol{p}_{\text{anchor}} = \boldsymbol{p}_c + k_{\text{offset}}\boldsymbol{v}_c$$

Assume that no characters are lagging such that the current center of mass is the position of the previous anchor position (i.e., k_{offset} is small enough that velocity of the center of mass is sufficiently smaller than the maximum velocity of the characters).

b) (3%) Now use your previous calculations to update the slot positions using the new calculated anchor point using the relative slot position as defined in the following equation,

$$\Delta \boldsymbol{p}_{s_i} = \boldsymbol{p}_{s_i} - \boldsymbol{p}_{anchor},$$

where $\boldsymbol{p}_{s_i}$ is the position of slot i. Note that the relative slot position is computed using the anchor position *before* the anchor position is updated (which, by the note at the end of a), is $\boldsymbol{p}_c$).

Question #3: (75%) [Practical Question]

Problem Statement:

For the question you are to write a C# program to run on the Unity Game Engine to have a collection of NPCs governed by the above behaviour heuristics play a simple game of Freeze Tag. Your program must comprise of the following:

R1) Environment and NPCs

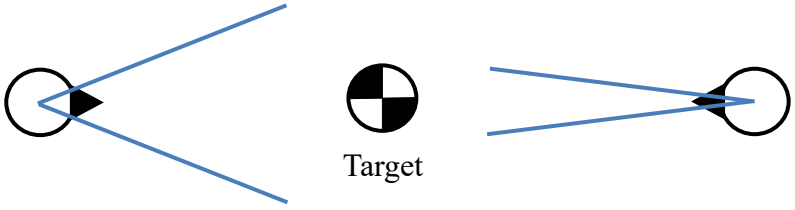
- Using Unity as your game engine, define an environment in the form of a 2½D toroidal arena. This is simply a 2D plane with no obstacles upon which will be placed 3D characters that remain upright and if a character goes off one side of the arena, then that character will reappear on the opposite side (just like the demo programs used by the [M&F] text that were shown in class). Make sure the (stationary) view point can see the entire arena.
- Populate the arena with 3D NPC characters (the number is up to you, but you probably would want at least four). Do not use 2D sprites. How interesting you make your 3D NPCs is up to you. As long as you can perform affine transformations (translate and rotate) on the character, and can tell which way it is pointing (e.g., texture one side of it with a smiley face, or give it a pointy nose), that will be minimally sufficient for this assignment. If you wish to use an animated character (e.g., with animations for standing and walking), that would be more interesting, but this should be a final step in your program development.

R2) Human Character Behaviour:

- The following are a set of heuristics widely used by game developers for the movement of human characters while *trying to reach* a target, based on the description given on page 176 of *Artificial Animation for Gaming, 2nd Ed.* Millington and Funge [M&F]:

A. If the character is stationary or moving very slowly then
   i. If it is a very small distance from its target, it will step there directly, even if this involves moving backward or sidestepping,
   ii. Else if the target is farther away, the character will first turn on the spot to face its target and then move forward to reach it.

B. Else if the character is moving with some speed then
   i. If the target is within a speed-dependent arc (like a narrow perception zone, or a cone of perception; see figure below; the higher the speed of the NPC, the narrower the cone) in front of it, then it will continue to move forward but add a rotational component to incrementally turn toward the target,
   ii. Else if the target is outside its arc, then it will stop moving and change direction on the spot before setting off once more.



Moving at moderate speed.

Moving at rapid speed.

The relative size of your environment and characters, as well as the number of characters you have, will affect the parameters mentioned above: the small radius for sidestepping; how fast is moving "very slowly;" and the relationship between the size of the speed-dependent arc and the velocity of the character.

For this assignment, we can also add heuristics for trying to *stay away* from a target:

C. Regardless of the speed of the character:
   i. If it is a very small distance from its target, it will step away directly, even if this involves moving backward or sidestepping,
   ii. Else if the target is farther away, the character will first come to a stop, turn on the spot to face away from the target, and then move away.

- Give each NPC the human character behaviour heuristics described above using the kinematic movements listed below.

Behaviour Implementation

Heuristic	Changing Position	Changing Orientation
A.i.	Kinematic *Arrive*	None
A.ii.	Kinematic *Arrive*	Interpolated Change in Orientation*
B.i.	Kinematic *Arrive*	Interpolated Change in Orientation
B.ii.	Kinematic *Arrive*	Interpolated Change in Orientation
C.i.	Kinematic *Flee*	None
C.ii.	Kinematic *Flee*	Interpolated Change in Orientation

* Match the orientation of the character with the direction of the velocity, interpolated over a fixed (small) number of frames.

Your implementation of the movement behaviours will follow that of [AI-M] so you will need to set a time to target ($t2t$) (which should be a relatively short period of time), radius of satisfaction, maximum velocity, etc. The C++ source code from [M&F] is available from `https://github.com/idmillington/aicore`

R3) A game of Freeze Tag:

- The game of freeze tag can be defined as follows. The freeze tag game begins with one character chosen at random to be the tagged character. The tagged character then chases the other characters, one at a time, until it touches one whereupon the latter is then frozen at its position. The tagged character continues chasing any unfrozen characters, freezing each in turn. Only the "pursued" untagged character should actively flee from the tagged character. Meanwhile, the unfrozen characters can "unfreeze" a frozen character by touching them. Once all the other characters except the tagged character have been frozen, the last character to be frozen takes over the role of the tagged character and the game continues.

- Engage your NPCs in a game of Freeze Tag. All the characters will use the human character behaviour described above with the following changes. The tagged character should use *Pursue* behaviour instead of *Arrive*. Kinematic *Pursue* is same as Steering *Pursue* except the target position is delegated to Kinematic *Seek*. To make the game more interesting, you may also set the maximum velocity of the tagged character to be a bit larger than that for the non-tagged characters.

- Which character the tagged character should pursue is up to you: the nearest one would seem to be the best choice but notice that the target character will change if another character strays closer than the target.

- For each of the unfrozen characters, if they are not being pursued by the tagged character, they will try to "unfreeze" any one of the frozen characters by moving to that

selected character and "tagging" it. Which one to unfreeze is again up to you: the nearest one, the furthest one, or a randomly chosen one. If there are no frozen characters, you decide what you want the untagged characters to do: *Wander*, stand still (i.e., do nothing), a mixture of behaviours, etc. The unfrozen characters not being pursued may have some awareness of the tagged character but should not be actively fleeing.

- Of course, some of the elements of this game still need to be defined more precisely. E.g., how does a character "touch" (or "tag") another character? You'll have to determine these. Some balance between the rate of freezing and unfreezing of characters (even for only a short period) would make the game more interesting. You should make it clear which character is the tagged character. Don't worry about collisions between characters.

EVALUATION CRITERIA For Question 3 of this assignment

1. Only working programs will get credit. The marker must be able to run your program if it works to evaluate it, so you must give any instructions necessary to get your program running. If your program does not run, we will not debug it.
2. Breakdown:
   - R1: 15% (equally divided among the requirements listed in item R1 above)
   - R2: 35% (equally divided among the requirements listed in item R2 above)
   - R3: 30% (equally divided among the requirements listed in item R3 above)
   - Richness of behaviours and general aesthetics : 10%
   - Source program structure and readability: 5%
   - Write-up: 5%

Note on marking: As designed, there can be a wide variation in fulfilling the requirements of this assignment; just barely meeting the requirements will not result in a full mark. For example, if your program only barely has the features listed in R2, do not expect full marks. Behaviours where there is an obvious extra effort made to make the interaction between NPCs more realistic/interesting/appealing may expect full or nearly full marks (this is the purpose of the 10% for "richness of behaviours").

What to hand in for Assignment

Questions 1-2: (25%) (Theoretical Questions)
- Submit your answers to questions 1 and 2 **electronically** (as a PDF file, naming it as **TheoryYourIDnumber.pdf)**. Submit this file under "Theory Assignment 1".

Question 3 (75%) (Practical Question)
Submission Deliverables (Only Electronic submissions accepted) :

1. Submit a well-commented C# program for Unity including data files, if any, along auxiliary files needed to quickly get your program running, and any other instructions for compiling/ building/running your program. The source codes (and brief write-up, explained below) must be submitted **electronically** in a **zip format** with all the required files (example: **YourIDnumber.zip**). Submit this zip file as "Programming Assignment 1". ***Please do not e-mail the submission.***
2. Demonstrate your working program from an **exe file dated on or before January 27 at 23h55** to the lab instructor in the lab during the period of January 28-30.
3. Included in your zip file will be a brief write-up about your program explaining the characteristics of the behaviours you've defined and any special features that you would like us to consider during the evaluation.