Dept. of Computer Science & Software Eng., Concordia University
COMP 476/6331 --- Winter 2021

Advanced Game Development

Assignment 1 --- Due Sunday, February 14, 2021

This assignment will give you the opportunity to learn more about programming AI behavior. 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: (13%) [Theoretical Question]

For the following, assume that the center of an AI character is $\boldsymbol{p}_c = (11, 2)$ and that it is moving with velocity $\boldsymbol{v}_c = (0,4)$. Also, assume that the stationary target is at location $\boldsymbol{p}_t = (6,6)$. Other parameters are the time between updates, $t = 0.5$, the maximum velocity, $v_m = 10$, and the maximum acceleration, $a_m = 24$. For a) to c), give some details as to how you compute the first position (detailed calculations are not needed for the rest of the positions).

a) (4%) 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) (2%) What are the positions of the AI character at the next five updates if we use (Kinematic) *Flee* the target?

c) (5%) What are the positions of the AI character at the next five updates if we use (steering) *Seek* to reach the target?

d) (2%) For both (Kinematic) *Seek* and (steering) *Seek*, describe the path followed by the AI character when trying to get to the target position.

Question #2: (12%) [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, 1)
2	(6, 13)	(5, 11)	(3, 3)
3	(29, 12)	(28, 9)	(6, 5)

a) (4%) First calculate 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) (4%) 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 $\boldsymbol{p}_c$).

c) (4%) What would be the effect on the anchor and slot positions if character 3 was killed just before the above calculations (for a) and b))?

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

Problem Statement:

For this question you are to write a C# program to run on Unity Game Engine, to have a collection of NPCs governed by the above behaviour heuristics play a simple game of Capture the Flag. 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.

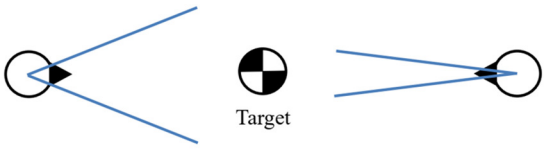
- Populate the arena with 3D NPC characters (the number is up to you, but you probably would want at least four). Do not use 2D sprites. How interesting you make your 3D NPCs is up to you. As long as you can perform affine transformations (translate and rotate) on the character, and can tell which way 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) 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.

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



  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 of the NPCs the human character behavior heuristics described above. Toggle (e.g., by pressing a key) between the following two implementations.

Behavior Implementation I

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.

Behavior Implementation II

Heuristic	Changing Position	Changing Orientation
A.i.	Steering Arrive	None
A.ii.	Steering Arrive	Face (delegated to Align)
B.i.	Steering Arrive	Looking Where You're Going (delegated to Align)
B.ii.	Steering Arrive	Face
C.i.	Steering Flee	Looking Where You're Going
C.ii.	Steering Evade	Face Away (delegated to Align)

The steering behavior *Face Away* is the opposite behavior to *Face* in that the character will be oriented to face away from the target.

Your implementation of the movement behaviours will follow that of [M&F] so you will need to set a time to arrive *t2t* (which should be a relatively short period of time), radius of satisfaction, maximum velocity, maximum acceleration, *etc.* To make the game more interesting, for implementation I (only) you may also set the maximum velocity of the tagged character to be a bit larger than that for the non-tagged characters. The C++ code from [M&F] is available from **https://github.com/idmillington/aicore**

R3) A game of Capture the flag:

- The game of "Capture the flag, commonly abbreviated as CTF, is a traditional outdoor game where two teams each have a flag (or other marker) and the object is to capture the other team's flag, located at the team's "base," and bring it safely back to their own base. Enemy players can be "tagged" by players in their home territory; these players are then, depending on the agreed rules, out of the game, members of the opposite team, sent back to their own territory, frozen in place until freed by a member of their own team, or 'in jail.'" [http://en.wikipedia.org/wiki/Capture_the_flag]

- Have your NPCs play Capture the flag. Divide your NPCs into two roughly evenly sized teams. Divide your arena into two areas and plant a flag in each area. To implement CTF, each team will randomly choose one of its NPC (from within the home area) to move to the enemy team's flag, capture it, and then move to the closest point (remember, this is a toroidal arena) in its home area. The first team to accomplish this is the winning team. If the chosen NPC is tagged, the flag is automatically returned to its original position and another NPC from the team is randomly chosen from its home area to capture the flag. Visually indicate where the flags are at all times.

- By default, the NPCs for a team will use *Wander* unless tasked with a specific job. For more interesting behavior, you may want to use *Separation* (for Implementation II), a mixture of behaviours, etc.

- Enemy NPCs are to be tagged by NPCs within their home areas. Tagged NPCs will be frozen. The closest NPC within its home area (that is not already trying to tag an enemy NPC) to an enemy NPC will try to tag it. Pursuit ends if the enemy NPC leaves the home area. While trying to tag an enemy NPC, the character should use *Pursue* behavior instead of *Arrive*. Note: Kinematic *Pursue* is same as Steering *Pursue* except the target position is delegated to Kinematic *Seek*.
- All tagged enemy NPCs are to be frozen in place until freed by a member of their own team. So, any NPC neither pursuing an enemy NPC nor trying to capture the enemy flag will seek out the closest frozen NPC from its team, if any, to unfreeze it.
- Of course, some of the elements of this game still need to be defined more precisely. E.g., how does a character "touch" another character? You'll have to determine these. 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 **on Moodle** (as a PDF file **TheoryYourIDnumber.pdf)**

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 code (and brief write-up, explained below) must be submitted on Moodle in a **zip format** with all the required files (ex.: **YourIDnumber.zip**). ***Please do not e-mail the submission.***
2. Demonstrate your working program from an **exe file dated on or before February 14 at 24h55** to the lab instructor in the lab during the period of February 15-19.
3. Included in your zip file will be a brief write-up about your program explaining the characteristics of the behaviors you have defined and any special features that you would like us to consider during the evaluation.