Assignment 3

CS 381: Game Engine Architecture
Spring 2019
Max Score: 100

Assignment

You will move towards oriented 2D physics while continuing handling multiple entities and tab selection to control how entities move realistically in your developing game engine. Please use the posted solution to assignment two as the basis for this assignment.

Multiple entities and selection (5)

Make a large ocean, a 10000 × 10000 planar textured surface located at the origin on the xz-plane at a height specified by a `surfaceHeight` variable. Select and load at least one example of each of five (5) different types of ship models (entity types) at `http://www.cse.unr.edu/~sushil/models/381/` into your evolving game engine. Bind the tab key to selection - that is - pressing the tab key will round-robin select the "next" entity. A selected entity, and only the selected entity, has a visible axis aligned bounding box.

You will notice that these entities are nautical vessels. Since boats and ships move on the surface of water, you will only need to deal with two-dimensional motion in our game engine's three dimensional world. Experiment with the different ways of texturing your ground plane with water. You may use the previous assignment's posted solution for pointers.

Oriented Vector Physics

Control

The arrow keys (or the numpad keys) on your keyboard control the selected entity. These keys control an entity's **desired** speed and **desired** heading. Given a desired speed and desired heading from user input, the entity's physics aspect changes an entity's actual speed and heading according to a specific entity's acceleration, a, and turn rate, tr. Once you have an actual speed and heading, you can use simple trigonometry to compute the entity's velocity vector. Given a velocity vector, an entity's position is simply `pos = pos + vel * dt` where **pos** and **vel** are 3D vectors.

When desired heading (dh) and desired speed (ds) change, the entity's actual speed (s) and actual heading (h) change as follows.

$$s = s \pm a * dt$$

Here, the $\pm$ denotes the two cases when ds is greater than and less than s. And, similarly

$$h = h \pm tr * dt$$

Note that you should limit the values of desired speed and speed to between 0 and some maximum. The same holds for desired heading and heading.

Physics

We will go over the trigonometry needed to compute a velocity vector from a scalar speed and velocity in class and in lab. Think of the speed, s, as the magnitude of this velocity vector while the heading, h, is the angle the vector makes with the positive x-axis. Consider

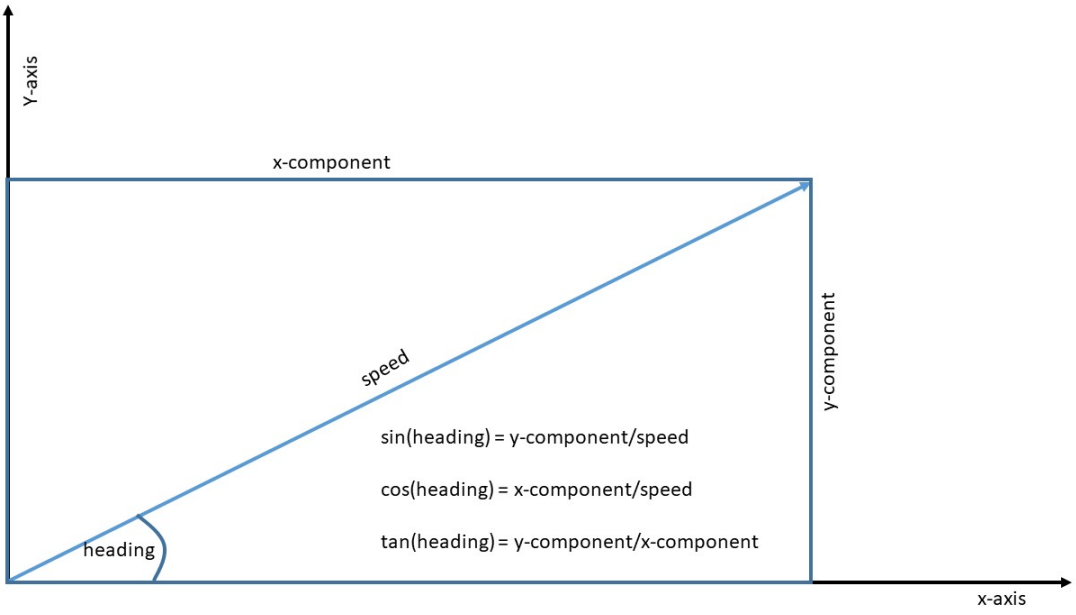

Figure 1: Trigonometry

Figure 1. In a 2D coordinate system with x, horizontal, and y, vertical, axes, we can see that given a heading and speed, we can get the x and y components of the corresponding 2D velocity vector from the following equations.

$$sin(h) = \frac{y}{s}$$

which means that the y-component of the velocity vector is

$$y = s \times sin(h)$$

and in the same way, given that

$$cos(h) = \frac{x}{s}$$

the x-component of the velocity vector is

$$x = s \times cos(h)$$

Thus the 2D vector would be $(s\,cos(h), s\,sin(h))$. You will derive and use a similar formula to compute a 3D velocity vector in our game world so that you can easily compute position with `pos = pos + vel * dt`. Note that ships travel on the water's surface, the xz-plane.

Overall, ships and boats must move realistically and make smooth turns. Large ships turn more slowly than small boats. Note however that although a slow turner, the aircraft carrier is the fastest ship in the ocean. Clearly, your physics code must now implement more realistic physics.

0.1 Camera control (5)

You will continue to control the camera with the WASD and R and F keys. In addition, you will now also use Left-Shift-A and Left-shift-D keys to control camera yaw and the Left-Shift-W and Left-Shift-S keys to control camera pitch.

0.2 Quitting

Hitting the escape key should gracefully shut down your running game engine.

0.3 Engine Architecture and Design (75)

Here are architecture and design elements that you MUST follow. I assume that your `_createScene` function is in `as3.cpp`

- (10 points) You will create one instance of an `EntityMgr` class in `_createScene`. The `EntityMgr` manages entities. This means the `EntityMgr`
  1. maintains a list (or map or array) of all entities in your game engine
  2. tracks which entity is currently selected
  3. creates entities and assigns them a unique identifier. This means you will need a `Entity381* CreateEntityOfTypeAtPositionAndHeading(EntityType type, Ogre::Vector pos, float heading)` method. I incompletely specify `Entity381` below.
  4. has a `void Tick(float dt)` method. This method iterates through the list of Entity381s and calls each Entity381's `Tick` method.

- (10 points) Create and use an `Entity381` class to hold information about your entities. Entity381 members must include

  1. entityId, entityName
  2. minSpeed, maxSpeed, speed, heading, desiredSpeed, desiredHeading
  3. acceleration, turningRate
  4. meshfile name
  5. position, velocity
  6. ogreSceneNode, ogreEntity (pointers to the entity's `Ogre::SceneNode` and `Ogre::Entity`)
  7. ...other items...
  8. a list of `Aspects`. We will specify Aspects below.
  9. Apart from the constructor, Entity381 will also declare and define a `void Tick(float dt)` method. This method will iterate through the list of this Entity381's Aspects and call each Aspect's Tick method. For this assignment you will create the two aspects listed and described below.
  10. other members as needed

- (10 points) Each of the five entity types (Destroyer (ddg51.mesh), Carrier (cvn68.mesh), Speedboat (cigarette.mesh), Frigate (sleek.mesh), Alien (alienship.mesh) will be a seperate subclass of `Entity381`. All these classes can be in (`Entity381.cpp and Entity381.h`) and do not need to be distributed one class per file. Note again that you must create one instance each of the above five types of Vessels at `http://www.cse.unr.edu/~sushil/models/381/`.

- Create and use an `Aspect` class and two subclasses of Aspect. The Aspect class is simply a base class to hold

  – a pointer to this aspect's Entity381

  – a virtual `void Tick(float dt)` method that will be overridden by subclasses.

  Each entity will have two aspects one of each subclass type below.

  1. Physics (40 points) - For an entity, if the entity's desired speed is not the entity's actual speed, the physics aspect changes the entity's speed using the entity's acceleration as the rate of change of speed. The same process applies to a selected entity's desired heading, heading, and turning rate. Once speed and heading are updated, you can compute the vector velocity. Now you can apply our old familiar code to update the position (`pos = pos + vel * dt`).
  2. Renderable (20 points) - Renderable manages the scene node (or nodes) and `Ogre::Entity` associated with our `Entity381s` and on every tick, copies the position **and heading** from our `Entity381` to the `Entity381`'s scene node.

- Please use Ogre's vector classes for vector math.
- In your `frameRenderingQueued` method you will call, your `EntityMgr`'s `Tick` method. This will go through the list of Entity381s and call each entity381's Tick method. Each Entity381's Tick method will then iterate through that Entity381's two Aspects and call each Aspect's Tick method.

These constraints will result in a clean, streamlined, design for your emerging game engine. They will also increase your understanding of oriented physics, inheritance, and game engine architecture.

Extra Credit

This is cumulative!

- Add mouse selection (+5)
- Add islands in your waterworld (+3)
- Find and use new 3D models. Hand in a document that provides a step by step tutorial on how to incorporate each of your models in the game engine.
- Add group mouse selection (+10)
- Add the ability to use standard RTS game mouse commands in order to have a selected (left-clicked) ship or boat to intercept another (right clicked) ship (+10)

Turning in your assignment

Assume that this format will be used for all your laboratory assignments throughout the semester unless otherwise specified.

1. Demonstrate your working program in the lab on the due date.
2. In lab, submit your code using Canvas.
   (a) Make a subdirectory in your home directory named `as3`.
   (b) Place all your project files in as3 (you will demo from this folder in lab).
   (c) Tar and gzip or Zip the entire folder and submit using Canvas. **If you do not submit this file, you will lose** 20% **points.**

Ask me (sushil@cse.unr.edu) if you have questions.

Objectives

1. Demonstrate an ability to apply knowledge of computing, mathematics, science, and engineering by learning and applying knowledge of Python to solve a problem (1)

2. Demonstrate an ability to analyze a problem, and identify, formulate and use the appropriate computing and engineering requirements for obtaining its solution (5)

3. Demonstrate an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (11)

4. Demonstrate an ability to apply design and development principles in the construction of software systems or computer systems of varying complexity (13)