Fall 2016
CS 529 | Fundamentals of Game Development
Project 1 | Asteroids

Files (submit folder) due

- **Part 2** – Monday, September 26th, 2016
- 11.55pm

Topics

The assignment will cover the following topics

1. Implement an "Asteroids" game, including:
    a. Building a 2D matrix library.
    b. Building a 2D collision library.
    c. Building a 2D vector library.
    d. Ship movement based on acceleration, velocity (and velocity cap).
    e. Asteroids movement based on velocity.
    f. Bullet spawning and movement based on velocity.
    g. Collision checking.
    h. Homing missile.

Goal

The goal of this assignment is to implement a 2D asteroids game, which will include matrix, vector and collision libraries, in addition to the implementation of "physics movement" which will be used to update the positions of the game object instances.

Assignment Submission

- Check the course syllabus regarding the naming and submission convention.

Description

I. Language: C/C++

II. A start-up application will be provided.

III. A library will be provided (For part 2), which includes several hardware related functions like initializing/updating and freeing the graphics and input engines.
   a. Library name: "Alpha_Engine.lib"/ "Alpha_Engine.dll"
   b. The header files of the "Alpha_Engine.lib"/ "Alpha_Engine.dll" library are included in the solution folder.

IV. The project is divided into 2 parts
   a. Part 1: Implement the vector, transformation and static intersection libraries
   b. Part 2: Implement the Asteroids game

Part 1

I. Implement the 2D vector library
   a. Function declarations are found in Vector2D.h
   b. Implement the functions in Vector2D.c
   c. Detailed explanations are found in the .h file

II. Implement the transformation library
   a. Function declarations are found in Matrix2D.h
   b. Implement the functions in Matrix2D.c
   c. Detailed explanations are found in the .h file

III. Implement the static intersection library
   a. Function declarations are found in Math2D.h
   b. Implement the function in Math2D.c
   c. Detailed explanations are found in the .h file

Part 2

I. Implement the Asteroids game.

II. The project can be completed without creating any additional files. But creating and adding files is allowed (For organization purposes).

III. Copy your matrix, vector and math .c and .h files to the solution folder and add them to the project.

IV. **Tutorial**: How to create a shape (mesh, which is a list of triangles). This will be needed to create the shapes of the bullet, asteroid and homing missile.

   a. Remember to create **normalized** shapes, which means all the vertices' coordinates should be in the [-0.5;0.5] range. Use the object instances' scale values to resize the shape.
   b. Call "AEGfxMeshStart" to inform the graphics manager that you are about the start sending triangles.

c. Call "AEGfxTriAdd" to add 1 triangle.
   - A triangle is formed by 3 vertices (points).
   - Create all the points between (-0.5, -0.5) and (0.5, 0.5), and use the object instance's scale to change the size.
   - Each point can have its own color.
   - The color format is: ARGB, where each 2 hexadecimal digits represent the value of the Alpha, Red, Green and Blue respectively. Note that alpha blending (Transparency) is not implemented.
   - Each point can have its own texture coordinate (set them to 0.0f in case you're not applying a texture).
   - An object (Shape) can have multiple triangles.
   - Call "AEGfxMeshEnd" to inform the graphics manager that you are done creating a mesh, which will return a pointer to the mesh you just created.

V. GameStateAsteroids.c

a. **Make sure to replace all the vector and matrix variables and functionalities by your own.**
   - Example: Replace AEVec2 by Vector2D, AEMtx33 by Matrix2D.
   - You can't use the AE's matrix, vector and collision functions. Use your own.

b. Get something to render on the screen!
   - TO DO 1: In the "GameStateAsteroidsUpdate" function, build the transformation matrix of each active game object instance.
   - The final transformation matrix of each game object instance should be saved in its transform component's "mTransform" member variable.

c. Implement the movement of the ship!
   - TO DO 2: In the "GameStateAsteroidsUpdate" function, update the position of each active game object instance.
   - TO DO 3: Compute the forward/backward acceleration of the ship when Up/Down is pressed.
     1. Then update its velocity.
     2. Then limit the maximum velocity, emulating friction.

d. Fire bullets!
   - TO DO 4: In the "GameStateAstreoidsLoad" function, create the bullet shape.
   - TO DO 5: In the "GameStateAsteroidsUpdate" function, create a bullet game object instance when "Space" is triggered.
   - TO DO 6: In the "GameStateAsteroidsUpdate" function, implement the bullet behavior.

e. Asteroids!
   - TO DO 7: In the "GameStateAstreoidsLoad" function, create the asteroid shape.

- TO DO 8: In the “GameStateAsteroidsInit” function, create 3 asteroid instances, each with a different size.
- In the “GameStateAsteroidsUpdate” function, implement the asteroid behavior (Check TO DO 6).

f. Collision!
- TO DO 9: In the “GameStateAsteroidsUpdate” function, implement the following collision tests:
  1. Asteroid to Bullet: Rectangle to Point check. If they collide, destroy both.
  2. Asteroid to Ship: Rectangle to Rectangle check. If they collide, destroy the asteroid, reset the ship position to the center of the screen.
- **Take the object instance's scale values into consideration when checking for collision.**

g. Homing missiles!
- TO DO 10: Create the homing missile shape.
- TO DO 11: In the “GameStateAsteroidsUpdate” function, create a homing missile game object instance when “M” is triggered.
- Implement the behavior of the homing missile (Check TO DO 6).
- The homing missile should always have a target (an asteroid), unless none exist. Each game loop, if it has no target, it should find one and try to follow it.
- It should rotate smoothly towards the target (Slower than “HOMING_MISSILE_ROT_SPEED” radian **per second**).
- If no target is available, the missile should move in a constant line, while still looking for a target once every game loop.
- Implement the Asteroid-Missile collision test (Check TO DO 9).

h. Free the level!
- TO DO 12: In the "GameStateAsteroidsFree" function, destroy all the active game object instances, using the “GameObjectInstanceDestroy” function.
- Reset the number of active game object instances (sgGameObjectInstanceNum).

i. Unload the level!
- TO DO 13: In the “GameStateAsteroidsUnload" function, destroy all the shapes (game objects), using the "AEGfxMeshFree" function.

Grading Part 2:

I. Ship movement:
   a. Incorrect acceleration: -15%
   b. Incorrect velocity: -15%
   c. Incorrect/missing velocity cap: -15%
II. No bullets, bullets don't move: -15%
III. No asteroids, asteroids don't move: -15%
IV. Incorrect/Missing collision: -30%
V. Missing/Incorrect homing missile: -10%
VI. Ship/Asteroids/Missile not warping on screen edges: -10%
VII. Crash/Freeze/Doesn't compile: -100%
VIII. No/Wrong submission: -100%
IX. Other possible mistakes: Up to -100%

Finally, each ".c", ".h", ".cpp" and ".hpp" file in your homework should include a header, which is included in the course syllabus.