DigiPen Institute of Technology
CSD1130
Game Implementation techniques
Assignment 2
Asteroids

Due date

Saturday Feb 04th, 2023, at 6:00pm

Topics

The assignment will cover the following topics.

1. Implementing Object Dynamics (acceleration and deceleration)
2. Apply Transformation Matrices
3. Runtime Object Creation
4. Implement Collision Intersection Between Animated Rectangles
5. Implementing Gameplay Aspects (scoring, game flow…)

Goal

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

Submission Guidelines

- To submit your programming assignment, organize a **folder** consisting of the following:
  - The solution "**.sln**" file
  - The project folder named "**CSD1130_Asteroids**"
  - Any other folder that your project depends on
- In other words, *your submission must be ready for compiling and linking by the grader*.
- Name this **folder** using the following convention, all small letters:
  - **<class>_<section>_<student login name>_<assignment#>**
  - For example, if your login is ***foo.boo*** and assignment 2 is being submitted, your **folder** would be named **csd1130_a_foo.boo_2**
- Your **folder** must not contain *Debug*, *Release*, *x64* and *x86* folders, *object files* or *executable* files.
- Your **folder** must not contain a **.db*, **.opendb*, **.sdf* or **.opensdf* files.
- Do delete the "**.vs**" read-only file. (*make sure windows explorer can show hidden files. ".vs" folders are usually set to hidden*). This will reduce the size of your submission file by a lot.
- Do delete the ".tmp" file as well.
- The provided project template is already organized in a way that source files and header files have their own folders, as well as a "Bin" folder (with 3 .dlls included) that will contain the generated output from "Release" or "Debug" versions. When submitting, make sure to **restore** the "Bin" folder to its **original** state.
- Zip this **folder** and name the resulting file using the following convention:
  <class>_<section>_<student login name>_<assignment#>.zip

  For example, if your login is ***foo.boo*** and you are submitting assignment 2, your zipped file would be named as: **csd1130_a_foo.boo_2.zip**

- Next, upload your zip file after logging into the course web page using the link https://distance3.sg.digipen.edu

- Finally, perform a **sanity** check to determine if your programming submission follows the guidelines by downloading the previously uploaded zip file, unzipping it, then compiling, linking, and executing your submission.

DigiPen Institute of Technology
CSD1130
Game Implementation techniques
Assignment 2
Asteroids

Using the right Compiler and MSVS setup

- The project must be tested in **RELEASE and DEBUG** modes with **warning level 4**, under **x64** platform. Under project settings the **SDK Version**: must be set to *10.0 (latest installed version)* and the **Platform Toolset** set to *Visual Studio 2019 (v142)*. It must generate 0 warnings and 0 errors. This can be verified on a PC located at **Pascal lab**.
- Please validate the previous statement before your submission!

Description

✓ A start-up application will be provided – Do open/use in **MSVS 2019**.

✓ Language: C (C++ “environment / setup only”).

✓ A library will be provided, which includes several hardware related functions like initializing, updating, and freeing the graphics and input engines.
- Library name: “Alpha_Engine”
- The header files and the lib files of the “Alpha_Engine” library are included in the solution folder.

Finally, each “.cpp” and “.h” file in your homework should include the following header:

```
/* Start Header *****************************************************************/
/*!
\file         <put file name here> (e.g. main.cpp)
\author       <provide your name, student login, and student id>
              (e.g. blabla blibli, bla.bli, 60001906)
\par          <provide your email address> (e.g. email: bla.bli\@digipen.edu)
\date         <date on which you created this file> (e.g. Jan 01, 20xx)
\brief        <provide a brief description of the file here>

Copyright (C) 20xx DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents
without the prior written consent of DigiPen Institute of
Technology is prohibited.
 */
/* End Header *******************************************************************/
```

Implementation

- ✓ GameStateAsteroids.h
  - No changes should be made to this file.

- ✓ GameStateList.h
  - No changes should be made to this file (*Except for Extra Credits*).

- ✓ GameStateMgr.h
  - No changes should be made to this file.

- ✓ GameStateMgr.cpp
  - Implement the “GS_ASTEROIDS” case in the “GameStateMgrUpdate” function by assigning the function pointers to the appropriate functions.

- ✓ GameStateAsteroids.cpp

  - In the “GameStateAstreoidsLoad” function:
    - ➢ Create the asteroid game object (shape)
    - ➢ Create the bullet game object (shape)
    - ➢ 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.

    - ➢ Call “AEGfxMeshStart” to inform the graphics manager that you are about to start sending triangles.
    - ➢ Call “AEGfxTriAdd” to add a triangle.
      - × A triangle is formed by 3 **counterclockwise** 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 hex digits’ represents the value of the Alpha, Red, Green and Blue, respectively.

  - In the “GameStateAsteroidsInit” function:
    - ➢ Create **4** initial asteroids instances with defined positions and velocities.
    - ➢ Each asteroid should have a different size, position, and velocity.

- In the "GameStateAsteroidsUpdate" function (more details are found in the asteroids template project under "GameStateAsteroidsUpdate" function):
  - Update the ship's acceleration/velocity/orientation according to user input.
    - AEInputCheckCurr: Checks pressed keys.
    - AEInputCheckTriggered: Checks triggered keys.

  - Create bullet instances by triggering the "Space" bar along with its direction and velocity.

  - Compute game object instances' starting bounding rectangles position. Update game object instances' positions according to their velocities.

  - Check for collision between active game objects.
    - Game instances will be encapsulated with bounding rectangles. Checking for collision between moving rectangles will require calling the "`CollisionIntersection_RectRect`" function.
    - The function prototype of "`CollisionIntersection_RectRect`" looks like this:

      -
        ```
        bool CollisionIntersection_RectRect (
        const AABB & aabb1,
        const AEVec2 & vel1,
        const AABB & aabb2,
        const AEVec2 & vel2);
        ```

      - The return type `bool` should return `0` for "No Intersection" and `1` for "Intersection".
      - The first argument "`aabb1`" of type "`AABB`" holds the information of the bounding box of the first object instance. The "`AABB`" type have the following structure:

        ```
        struct AABB
        {
        AEVec2 min;// holds the minimum point of the box
        AEVec2 max;// holds the maximum point of the box
        };
        ```

      - The second argument is the velocity of the object instance.
      - As for the next two arguments "`aabb2`" and "`vel2`" have the same properties as "`aabb1`" and "`vel1`" respectively, but they handle the information of the second bounding box.

        - Note: A more detailed explanation and a pseudo-code is in "Collision.cpp" file and in the uploaded lecture slides

    - Update specific game object instances' behavior.
        - Wrapping the asteroids and the ship around the world is required.
        - Removing the bullets when they go out of bounds is required.

    - Calculate the transformation matrix of each active object instance.
        - Remember that the order of matrix concatenation is important!
        - Order of matrix concatenation: Translation*Rotation*Scaling

- In the "GameStateAsteroidsDraw" function, do the following for each game object instance:
    - Set the transformation matrix of each game object instance using the "AEGfxSetTransform".
        - Typecast your Mtx33 transform matrix to AEMtx33 when passing it to the "AEGfxSetTransform" function (In case you implemented your own matrix library. Otherwise, you can skip this step).
    - Draw the object instance using the "AEGfxMeshDraw" function.
    - We will be using "printf" function to display score and lives values in the Window console. Display "*You Rock*" when you win (`sScore >= 5000`).

- In the "GameStateAsteroidsFree" function:
    - Kill each **active** game object instance using the "gameObjInstDestroy" function.

- In the "GameStateAsteroidsUnload" function:
    - Free each **used** game object (shape) using the "AEGfxMeshFree" function.
- **Do not change the GameObj struct and the GameObjInst struct unless you get the teacher's approval.**

DigiPen Institute of Technology
CSD1130
Game Implementation techniques
Assignment 2
Asteroids

Game Requirements

The following are the requirements that you must implement for this assignment:

- Infinite number of asteroids. Keep creating asteroids as long as the ship is still alive. Asteroid creation and movement should not be predictive.
- Scoring system.
- The ship has at least three lives.
- Win/lose conditions.
- Game well balanced and FUN!!!

Extra Credits

You are encouraged to implement extra features and here are some of the ideas:

- Special weapons
- Crates that give shield for the ship, or speed etc.
- Particle system (explosion, fire, etc.)
- Menu Level (Play/Exit/…)
- High score
- All variables displayed using "printf" are also available in "Text objects" inside the game.
- Add textures.
- Etc.

Basically, anything that it is worth showing! You will get extra letter grades according to the difficulty of the extra work.

A "**Resources**" folder is included near the ".sln" file. In case you would like to use the font system, or you would like to add some textures, the files must be in this folder.

Add an "**ExtraCredits.txt**" file next to the solution (.sln) file, if you have any, where you explain the feature(s). Otherwise, the extra credits are **not** counted!

Evaluation

Here are the most common reasons assignments are marked down:

- Project does not build.
- Project does not build without warnings.
- One or more items in the "Requirements" section was not satisfied.
- A fundamental concept was not understood.
- Code is sloppy and hard to read (e.g., indentation is not consistent, no comments, etc.)
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, etc.
- Project assignment was turned in late.

Grading Algorithm

This project will be graded according to the following rules. Any rule that is missing, letter grades will be deducted from the assignment's total grade:

- Not following the submission guidelines (from above section)
- Missing header comment, at the top of the modified/added only ".h" and ".cpp" files.
    - For any amount of files
    - Penalty applies, only, on the files that the students update or add.
- Header comment does not match the given one or missing essential information as described under "Documentation" section.
    - For any amount of files
    - Penalty applies, only, on the files that the students update or add.
- Compile errors or does not compile for some reasons.
- The demo is crashing at runtime:
    - Demo crashes right from the beginning
    - Demo crashes while playing, or at random places.
- Compile Warnings (Unique Instances)
- Output not matching the given one:
    - Rectangle-Rectangle collision detection not working with a major mistake.
        - Or the collision is buggy and is partially working.
    - Implementation of object dynamics (velocity, acceleration, deceleration) not working
        - Or the velocity, acceleration and/or deceleration is buggy.
    - No runtime object creation (bullets and/or asteroids)
    - No game play aspects (score, game flow, …)
        - Or partial game play aspects
    - Game not balanced (too easy or too hard)
- Your Output gameplay must mimic the original "**Example Demo**" provided. Any new "Extra Credits" gameplay must add on top of the original gameplay behavior, and not replace it! Otherwise, you may lose points!
- Matrices multiplication order is wrong.
- The frame rate is so low that it is barely playable.
- Checking the source code:
    - No code comments.
    - No consistent indentation.
    - No appropriate identifiers (proper variables naming).

- Extra Credits:
    - No points will be given for extra credits if you do not add "**ExtraCredits.txt**" file, or wrongly named the file, or your submission is beyond the deadline.
        - One or two valid extra credits (you get partial extra credits)
        - Three or more valid extra credits (you get full extra credits)
- If having Memory leaks!
    - At the beginning of the "WinMain" function there are some commented code:

```
//// Enable run-time memory check for debug builds.
//#if defined(DEBUG) | defined(_DEBUG)
//   _CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
//#endif

//int * pi = new int;
```

Un-comment the code, select "**Debug**" mode, Press **F5** (or Start Debugging). After the program exists, you get a message in the MSVS "**Output**" window: (Detected memory leaks!). This is due for not deleting "pi" variable before exiting. Therefore, a memory is still allocated and not properly released.

(int * pi…), is just a sample for you to understand. Simply remove/delete this line when you start your work.

This is the same, if you are using "malloc" or "alloc" functions, you must call "free" function appropriately!

Notes

- Depending on different implementation you may want to add new "headers" or "cpp" files. You can add new code files by respecting the project folder's organization.
- Feel free to add textures if you would like to (under "Resources" folder).
- Do **NOT** change the "C" structure of the assignment to "C++".
- This assignment will fortify the **individual** student's knowledge about:
    - Using physics along with collision detection.
    - The usage of Alpha_Engine, which has a benefit on the **CSD1450** project.
    - How to handle objects and different instances.
    - A better project organization.
    - Avoiding memory leaks by writing cleaner code!
- The assignment may take on average 6 hours to finish.