DigiPen Institute of Technology
CSD1130
Game Implementation techniques
Assignment 3 – Part 2
Platformer

Due date
Saturday, March 04th, 2023 at 6:00pm

Topics
The assignment will cover the following topics:

1. Implement a “platform” game including:
    a. Binary collision
    b. Importing data from an editor (text file)
    c. Jump
    d. State machine
    e. Particle system

Goal
The goal of this assignment is to implement a 2D platform game. The level data will be imported from a text file (which was previously exported using a map editor).
Jumping will be based on gravity and velocity, while a state machine will be used to determine some sprites' behavior.

Submission Guidelines (check Grading Algorithm section)

- To submit your programming assignment, organize a **folder** consisting of the following:
  - The solution **.sln** file
  - The project folder named "**CSD1130_Platformer**"
  - Any other folder that your project depends on to build and run
- 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#>_<part#>**
  - For example, if your login is ***foo.boo*** and assignment 1 part 1 is being submitted, your **folder** would be named **csd1130_a_foo.boo_1_1**
- Your **folder** must not contain *Debug*, *Release, x64* and *x86* folders, *object files* or *executable* files.
- Your **folder** must not contain the **.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.
- 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#>_<part#>.zip

  For example, if your login is ***foo.boo*** and you are submitting assignment 1 part 1, your zipped file would be named as: **csd1130_a_foo.boo_1_1.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 3 – Part 2
Platformer

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 **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.

✓ One flow chart is provided:
  - The state machine that controls enemy characters.

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

```
/* Start Header **************************************************************/
/*!
\file         <put file name here> (e.g. main.cpp)
\author       <provide your name, student login, and student id>
              (e.g. DigiPen Singapore, dSingapore, 60001906)
\par          <provide your email address> (e.g. email: digipen\@digipen.edu)
\date         <date on which you created this file> (e.g. Mar 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

- ✓ Collision.cpp
  - Rectangle – Rectangle collision detection already implemented in Project 2.

- ✓ GameState_Platform.cpp
  - Add part1's functions to this file:
    - ➢ int GetCellValue(int X, int Y);
    - ➢ int CheckInstanceBinaryMapCollision(float PosX, float PosY, float scaleX, float scaleY);
    - ➢ void SnapToCell(float *Coordinate);
    - ➢ int ImportMapDataFromFile(char *FileName);
    - ➢ void FreeMapData(void);

  - Implement the enemy's state machine
    - ➢ void EnemyStateMachine(GameObjInst *pInst);
    - ➢ This state machine has 2 states: Going left and going right
    - ➢ Each state has 3 inner states:
      - × On Enter
      - × On Update
      - × On Exit
    - ➢ 2 enumerations are used for this state machine

```
//State machine states
enum STATE
{
        STATE_NONE,
        STATE_GOING_LEFT,
        STATE_GOING_RIGHT
};

//State machine inner states
enum INNER_STATE
{
        INNER_STATE_ON_ENTER,
        INNER_STATE_ON_UPDATE,
        INNER_STATE_ON_EXIT
};
```

    - ➢ Check the comment in the provided template and the provided chart.

- In the "GameStatePlatformLoad" function:
  - Compute "MapTransform" at the end of the function.
  - This matrix will be used later, when rendering object instances, to transform them from the normalized coordinates system of the binary map.

- In the “GameStatePlatformInit” function:
  - The black/white instances are already created. They will be used to draw collision and non-collision cells.
  - Loop through the elements of the 2D array "MapData” and create object instances according to the value of each cell.
  - Possible object instances to create:
    - Hero
    - Enemy
    - Coin

- In the “GameStatePlatformUpdate” function:
  - Update velocity X of the hero according to user's input.
  - Apply a jump motion in case the user pressed jump while the hero is on a platform.
    - The hero is considered on a platform if its bottom collision flag is set to 1.
    - Check pressed keys.

  - Update active object instances and general behavior.
    - Apply gravity to “hero” and “enemies” object instances only, using:
      Velocity Y = Gravity * deltaTime + Velocity Y
    - If the object instance is an enemy, update its behavior using the state machine "EnemyStateMachine”

  - Update game object instances' positions according to their velocities and their bounding rectangles.
    - Position = Velocity * deltaTime + Position
    - Update bounding rectangles

- ➢ Check for collision between the grid and the active game object instances
  - Update the collision flag of game object instances by calling the "CheckInstanceBinaryMapCollision" function.
  - Snap the position of the colliding object instances in case they were colliding from one or more sides.

- ➢ Check for collision between active and collidable game object instances
  - Collision check is basically be hero-coin or hero-enemy.
  - Loop through active and collidable object instances.
  - If it is an enemy, check for collision with the hero as rectangle-rectangle. Update game behavior accordingly (check comments).
  - If it is a coin, check for collision with the hero as rectangle-rectangle. Update game behavior accordingly (check comments).

- ➢ Calculate the transformation matrix of each active object instance.
  - Remember that the order of matrix concatenation is important!

- ➢ For level2:
  - Update Camera position and clamp it at the level's border.
  - You will need to implement a wall-jump to be able to reach the higher platforms and clear the level. Feel free to mimic the sample demo double-jump, or to create your own version. Do note that mid-air double jump is not allowed!
    - Comment your code, explaining how the wall-jump works.

- In the "GameStatePlatformDraw" function, we must draw the grid and the active and visible object instances.
  - ➢ Draw the grid
    - Loop through the width and height of the binary map.
    - Compute the translation matrix of each cell depending on its X and Y coordinates.
    - Concatenate the result with "MapTransform".
    - Send the resultant matrix to the graphics manager using "AEGfxSetTransform"
    - Draw "BlackInstance" or "WhiteInstance" depending on the cell's value.

- ➢ Draw the active and visible object instances
  - Concatenate the object instance's transformation matrix with "Maptransform"
  - Send the resultant matrix to the graphics manager using "AEGfxSetTransform"
  - Draw the object's shape using "AEGfxMeshDraw"
- ➢ Display “HeroLives” and “TotalCoins” values (in game Window).

- In the "GameStatePlaformFree" function:
  - ➢ Kill each game object instance using the "gameObjInstDestroy" function.
- In the "GameStatePlatformUnload" function:
  - ➢ Free the map data

Note:

- Read the comment in the project because it provides more details about the implementation of each aspect of the project.
- Starting the project for the first time, the game will exit after the first loop because “ImportMapDataFromFile” function is returning 0.
- For level2, you must use the “Exported2.txt” file that you can find in “Resources” folder.
- You may want to add “GameState_Platform2.h” and “GameState_Platform2.cpp” for level2. Or some students would like to combine both levels in the same code files.
- In Alpha Engine, you can’t add a font for each level, separately. You must add your font(s) only one time for all your levels, and free them just before you exit the application. In our assignment, I added some comments, in the “main.cpp” file, for the place where you can add/destroy a font asset.

Game Requirements

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

- At least 2 levels. Level1 should be as big as the viewport, as for level2 you need to create the level bigger than the viewport. In level2 you need to move the camera since the world is larger than the viewport. You must use "Exported2.txt" file for level2, found in the "Resources" folder of the provided sample output.
- Main menu state where you can choose which level to play or to exit.
- One particle system effect (applied on the main character)
- One enemy behavior (the one given in the state machine)
- Game well balanced and FUN!!! ☺

<u>**Evaluation**</u>

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 project's grade:

- Not following the submission guidelines (from above section)
- Missing header comment, at the top of the ".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.
  - 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 of a level
  - Demo crashes while playing, or at random places.
- Compile Warnings (Unique Instances)
- The AABB algorithm is not fixed from assignment 2
- Output not matching the "Sample Output" given one:
  - No binary collision detection or buggy binary collision detection
  - No particle system or a non-proper functioning particle system
    Note: You may implement a different particle system style.
  - No implementation of object dynamics – jumping, or buggy jumping implementation
  - No AI
    - Or buggy AI
  - No game play aspects (levels win/loose progress, lives, etc…)
- Wall-jump not implemented for level 2
  - Can mimic the sample-demo, or your own customized wall-jump.
  - If you customized it, you must write comments explaining it at your code location, in visual studio.
- Checking the source code:
  - No comments or functions documentation
  - No consistent indentation
  - No appropriate identifiers
- Memory leaks!
  - Code to check memory leaks was given in assignment 2.

- No Level2 or using a different map than the one provided.
- Buggy dynamic camera in level 2. Examples: does not follow the player or does not clamp exactly to the level's borders.
- Background map output is flipped, while the dynamic objects are at correct positions.
- No main menu.
- Gravity must be applied on enemies and hero (player) instances only!

Notes

- Depending on different implementation you might need to add new headers or .cpp files. You can add new code files by respecting the project folder organization.
- Do **NOT** change the "C" structure of the assignment to "C++".
- Your gameplay must mimic the gameplay of the "CSD1130_Platformer.exe" demo provided.