Summer 2013
CS 529 | Fundamentals of Game Development
Project 2 | Part 2 – Platformer - Particle System

Files (submit folder) due

- Without Particle Effects:
  - Tuesday, October 8th, 2013
  - 11.55pm

Topics

1. The assignment will cover the following topics.
   a. Implement a "platformer" game including:
   b. Binary collision.
   c. Importing data from an editor.
   d. Circle – Rectangle Collision.
   e. Jump.
   f. State Machine.
   g. Particle System.

Goal

- The goal of this assignment is to implement a 2D platformer game, which will include the previously implemented matrix, vector and collision libraries, in addition to some new functions like the "Circle-Rectangle" collision check.
- 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 used to determine some sprites' behavior.
- Particle systems will be implemented (At least 2, should be drastically different, chosen by the student).

Assignment Submission

- Compress (.zip) the solution folder (Delete the debug/release folders and the .ncb file first), and submit it on distance.digipen.edu.

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

Description

I. Implement the platformer game

II. Language: C

III. A start-up application will be provided.

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

V. One flow chart is provided:
   a. The state machine that controls enemy characters.

VI. Copy your Math2D, Vector2D, Matrix2D functions from previous projects.

VII. Implement the StaticCircleToStaticRectangle intersection function in Math2D.c

VIII. In GameStatePlatformer.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..

   b. 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);`

   c. Implement the enemy’s state machine
      - `void EnemyStateMachine(GameObjInst *pInst);`
      - This state machine ahs 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.

d. In the "GameStatePlatformLoad" function:

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

e. 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.

f. 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.
    - ▪ AEInputCheckCurr: Checks pressed keys

- Update game object instances' positions according to their velocities.

- Update active object instances and general behavior.
    - ▪ Apply gravity to all object instances using Velocity Y = Gravity * time + Velocity Y
    - ▪ If the object instance is an enemy, update its behavior using the state machine "EnemyStateMachine”

- Update the positions of active object instances
    - Position = Velocity * time + Position

- 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 game object instances
    - Collision check is basically between hero-coin or hero-enemy.
    - Loop through active object instances.
    - If it's an enemy, check for collision with the hero as rectangle-rectangle. Update game behavior accordingly (check comment).
    - If it's a coin, check for collision with the hero as circle-rectangle. Update game behavior accordingly (check comment).

- 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

g. In the "GameStatePlatformDraw" function, we must draw the grid and the active 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"
    - Draw "BlackInstance" or "WhiteInstance" depending on the cell's value.

  - Draw the active 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 "AEGfxTriDraw"

h. "AEGfxPrint" can be used to print a null terminated string on the screen.

i. In the "GameStatePlaformFree" function:
  - Kill each game object instance using the "gameObjInstDestroy" function.

j. In the "GameStatePlatformUnload" function:
  - Free the map data

k. Implement at least 2 particle systems

- Every particle system should be implemented in 3 steps: Create the particle system, Update the particle system, Update the particles.
- You can add members to the "GameObjInst" structure.
- Example: A particle system that occurs when the main character intersects with a wall or a platform.
    - Create the particle system when the intersection is detected, with a certain number of particles. The particles initial positions and velocities should depend on the collision side of the main character.
    - Update the particle system: No particles are generated besides the ones that were created initially.
    - Update the particles: Apply gravity and/or collision. Check the life counter in order to determine if the particle should be deleted.

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

```
/ *---------------------------------------------------------------------------
Project Title          :       Platformer
File Name              :       (Enter file name here)
Author                 :       (Enter your name here)
Creation Date          :       (Enter the creation date of the file)
Purpose                :       (Enter the main purpose of the file here)
History
 -(Enter date here)    :       (Enter modifications done on current date here)
 -(Enter date here)    :       (Enter modifications done on current date here)
-----------------------------------------------------------------------------*/
```