DigiPen Institute of Technology
CS230
Game Implementation techniques
Project 3 – Part 2
Platform

Due date: Tuesday 31st of March 2009

Topics
The assignment will cover the following topics

1. Implement a "platform" game including:
    a. Binary collision
    b. Importing data from an editor
    c. Circle – Rectangle Collision
    d. Jump
    e. State Machine

Goal
The goal of this assignment is to implement a 2D platform 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.

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

Your submitted assignment should use the following naming convention:
"username.zip" where ID represents your DigiPen account name.

Example: achacra@digipen.edu

Description

- ✓ A start-up application will be provided.
- ✓ Language: C (although the files' extensions are .cpp, which means you cannot use classes).
- ✓ 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.lib"

- The header files of the "Alpha_Engine.lib" library are included in the solution folder.

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

✓ No files other than the Matrix, Collision and Vector ones (.cpp & .h) should be created nor added to the project.

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

```
/ *-----------------------------------------------------------------------------
Project Title           :   CS 230: Project 3 Part 2 Platform
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)
-----------------------------------------------------------------------------*/
```

Implementation

✓ Copy your matrix, vector and collision .cpp and .h files to the solution folder.

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

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

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

✓ Main.h
- Include "Vector2.h"
- Include "Matrix3x3.h"
- Include "Collision2D.h"

✓ Collision2D.h and Collision2D.cpp
- Implement the circle-rectangle collision check function
  bool AETestCircleToRect (Vec2* pCtr0, float radius0, Vec2* pRect0, float sizeX0, float sizeY0);
- pCtr0: Center of the circle
- radius0: Radius of the circle
- pRect0: center of the rectangle

- sizeX0: Width of the rectangle
- sizeY0: Height of the rectangle
- This function returns 1 if there is collision between the circle and the rectangle, otherwise it returns 0.

✓ GameState_Platform.cpp

- Make sure to replace the library's structure with your own (Replace AEVec2 with Vec2, AEMtx33 with Mtx3...)

- 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);
    - **You can use static arrays for this function**
  - void FreeMapData(void);

- 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.

- 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.

- 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.
    - 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 and collidable game object instances
    - Collision check is basically be hero-coin or hero-enemy.
    - Loop through active and collidable 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

- In the "GameStatePlatformDraw" function, we must draw the grid and the active and visible object instances.
  - Draw the grid
    - Loop thourgh 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 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 "AEGfxTriDraw"

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

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

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