Fall 2016
CS 529 | Fundamentals of Game Development
Project 2 | Part 2 – Platformer - Particle System

Files (submit folder) due

- Part 2
  - Monday, October 10th, 2016
  - 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

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

Description

I. Implement the Platformer game

II. Language: C/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 with your own.
      - Example: Replace AEVec2 by Vector2D, AEMtx33 by Matrix2D..

   b. Import part 1's functions!
      - TO DO 1: Import your functions from part 1 (From BinaryMap.c & BinaryMap.h). You can copy the functions to "GameStatePlatformer.c ", or simply, include these 2 files in the project.

   c. Create the level! (Map & Instances)
      - TO DO 2: In the "GameStatePlatformInit", create the map tiles and initial game object instances.
      - Refer to the "OBJECT_TYPE" enumeration for objects' IDs.
      - If implemented correctly, you should see the map and instances. (They'll be really small at this point).

   d. Implement the map transform!
      - TO DO 3: In the "GameStatePlatformLoad" function, Implement the map transform matrix, and store the result in the "MapTransform" matrix variable.
      - To DO 4: In the "GameStatePlatformDraw" function, concatenate the "MapTransform" matrix with the transformation of each tile and game object instance before drawing it.

   e. Movement & Gravity!
      - TO DO 5: In the "GameStatePlatformUpdate" function, update the positions of each active game object instance.

- TO DO 6: In the “GameStatePlatformUpdate” function, update the velocity of each active game object instance that has a “Physics” component, using “GRAVITY”. The hero, enemies and coins should fall down at this point.

f. Map Collision!
   - TO DO 7: In the “GameStatePlatformUpdate” function, check for collision between active game object instances that have a "Component_CollisionWithMap" component and the map, by calling the “CheckInstanceBinaryMapCollision”.
     - Store the bit field in the instance's Component_CollisionWithMap.
     - In case of an intersection with the map, set the respective velocity coordinate to 0 and snap the respective position coordinate.
   - If implemented correctly, object instances should stop falling down through the platforms.

g. Input & Jump!
   - TO DO 8: In the “GameStatePlatformUpdate” function, implement the hero’s movement control.
     - If Left/Right are pressed: Set the hero velocity's X coordinate to -/+ MOVE_VELOCITY_HERO.
     - Set it to 0 when neither are pressed.
     - If SPACE is pressed AND the hero is on a platform: Jump. Use "JUMP_VELOCITY" as the upward jump velocity.
   - You should be able to move the main character and make it jump.

h. AI Movement!
   - TO DO 9: In the “void EnemyStateMachine(GameObjInst *pInst)” function, Implement the enemy’s state machine.
   - Each enemy’s current movement status is controlled by its “state”, “innerState” and “counter” member variables.
     - Refer to the provided enemy movement flowchart.
   - TO DO 10: In the “GameStatePlatformUpdate” function: If the object instance is an enemy, update its state machine by calling the "EnemyStateMachine" function.

i. Object to Object Collision
   - TO DO 11: In the “GameStatePlatformUpdate” function, check for collision among active game object instances.
     - Hero-Enemy intersection: Rectangle-Rectangle
     - Hero-Coin intersection: Rectangle-Circle

j. 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).

k. Unload the level!
- TO DO 13: In the "GameStatePlatformUnload" function:
  - Destroy all the shapes (game objects), using the "AEGfxMeshFree" function.
  - Free the map data.

l. Particle effects:
Implement at least 2 substantially different particle systems
- It's up to you to add new shapes, components, component specific behavior etc…
- Particle systems should be implemented in 3 steps:
  - Create the particle system.
  - Update the particle system.
  - Update the particles.
- 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. Grading

a. Incorrect Map Drawing" -20%
b. No "MapTransform": -15%
c. No collision with map: -25%
d. No main character jump or implemented incorrectly: -15%
e. No snapping: -15%
f. No velocity reset after collision with map: -15%
g. No object to object collision: -15%
h. No enemy state machine or implemented incorrectly: -20%
i. Less than average looking particle effect or missing particle effect: -15% each.
j. Remotely similar particle effects: Only one will count.
k. Crash/Freeze/Doesn't compile: -100%
l. No/Wrong submission: -100%
m. Other possible mistake: Up to -100%

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