Random Engine

by

Atanas Slavov, Felipe Fragoso and Sergio Valderrama

Developed for the Game Engines course

MsC Games, IT University of Copenhagen, Autumn 2016

Table of contents

I. **Engine structure and Class breakdown**
II. **Performance**
III. **Possible improvements and comments**
IV. **Demos and Game**
V. **Engine build notes**
VI. **Responsibility table**

Random Engine is a 2D game engine that wouldn't have been possible without Simple Render Engine, Box2D, SDL2, SDL2_Image, Glew, GLM, Picojson and CMake. Thanks to those external libraries we were able to focus on developing features and improving our engine. We'd also like to thank our teachers and teachers' assistants for their invaluable help during the development process.

I. Engine structure and Class breakdown

Random Engine utilizes a Scene Manager to create a vector of Scenes and then load their resources when needed. When a new Scene needs to be loaded, the engine and physics simulation are paused until the objects in the current Scene are destroyed, before the new Scene's resources are loaded. This way we ensure clean transitions between Scenes. Demo03 showcases our Scene Manager.

Scenes are made up of GameObjects that can be added through code or from a json file. We do the latter by using a modified version of the json parser from the exercises, found in GameObjectParser. The two approaches for adding Scenes are exemplified in Demo01 and Demo02.

The GameObjects themselves are Component based, making them highly customizable and powerful tools to build our Scenes with. The functions of the GameObject class are very much focused on working with Components. Aside from functions for adding and removing Components, the draw() function looks for any Component that needs to be drawn on screen (Sprite, Animation, Particle, Mesh), and makes the appropriate calls. The updateCollider() and updateGlobalTransform() functions assure synchronization of transform and collider for objects that use both, and the scriptStart(), scriptUpdate() and showGUI() functions round out the list by calling their mirrors if the object has a Scriptable component.

We have quite a few Component classes that inherit from the base Component. They are all children of the GameObject class. In total we have 7 components - Animation, Body2D, Mesh, ParticleEmitter, Scriptable, SpriteRenderer and Transform. In the following paragraphs we will go over what they all do.

The Animation component is, not surprisingly, responsible for playing animations. It does so by taking a vector of sprites to animate and the length of the animation in milliseconds in the constructor. The animation can then be played, paused or stopped, and the duration of the current animation can be changed with the corresponding functions. Demo01 showcases some of the features of the Animation component.

The Body2D component is responsible for adding colliders to objects. Random Engine has support for simple box and circle colliders. The component works with our Physics2D class, which has support for the Scriptable component's onCollisionEnter() and onCollisionExit() functions. Demo02 showcases the Body2D component.

As a side note, Random Engine has a debug mode, used for drawing the colliders through the DebugDraw class. They can be turned on and off for the demos within the GUI, and they are by default turned off for the game as they are only meant for debugging purposes.

The Mesh component is a legacy component for drawing 3D objects. It works but right now it has no effective purpose in the engine, and isn't used in the game or any of the demos.

ParticleEmitter is our particles component. The core structure hasn't changed overly much since the exercises; however, we can now set the color and velocity of the particles and use the Interpolation class for Linear, Bezier and "Random" interpolation between colors. Demo04 shows the options available.

Scriptable very much copies Unity's way of doing scripts, by allowing for function execution once at Start(), continuously on Update() and OnGUI(), and situationally with OnCollisionEnter() and OnCollisionExit(). The Scriptable component is in use throughout the demos and the game.

SpriteRenderer is of course responsible for rendering Sprites on screen. It is closely tied to the Sprite class, which is where the sprites are first created. The Sprite class itself is tied to the SpriteAtlas class, which parses json files. While the structure of the parser itself is largely the same as the one from the exercise, we made it so multiple json files can be parsed and added to the atlas' map for each scene. The SpriteRenderer is showcased in Demo01, Demo02 and the game. Using multiple jsons for the atlas can be seen in the game scene.

Transform is the component responsible for position, rotation and scale of objects. We can properly calculate the global transform of child objects, but global rotation proved tricky, so the relevant code was left commented out for the final version. The transform component is used to ensure the correct functionality of the SpriteRenderer, Animation, Mesh, ParticleEmitter and Body2D components inside GameObject's functions.

We added a Material class for our needs. It comes into use whenever we need to draw objects on the screen, so naturally we have materials for Sprites and Particles. There is also the Default material, and the AlphaBlend material, which uses a custom shader. The AlphaBlend material is on display in Demo01, the Particle material in Demo04, and the Sprite material is used everywhere else where we have Sprites.

Random Engine has its' own Input class, that adds keyboard keys to a map when they are first pressed, and then keeps track of their state. The class has functions that copy Unity's key state functions for getKeyDown, getKey and getKeyUp.

Finally, we have a GUI class that has been extended since the exercise. It can be seen in action throughout the demos and the game.

II. Performance

We think that performance is high up on the list for a games engine, so we spent a lot of time looking into memory leaks. When we started to merge all the components into our engine we were sure that we were not deleting correctly, so we implemented memory leak detection. What we did is overload the new and delete operators and check for the number of dynamic allocations and deallocations of memory. Since we added the Profiler, every time a new feature is introduced we check for leaks, to keep track of memory and work clean. We have a margin for memory leaks which are not related to Random Engine's memory uses and are out of our control.

DebugDraw is a feature used to show the colliders of GameObjects when testing the Scene. There is a noticeable drop in performance when there are a lot of objects in the scene and DebugDraw is enabled. This is examplified in Demo02 if you create a lot of plant objects. Turning off DebugDraw through the GUI noticeably improves performance.

III. Possible improvements and comments

Improvements

1) Implement multithreading for particles
2) Custom memory leak detection, to keep better track of what components are leaking memory
3) Export scenes to json file
4) SceneManager::loadScene() support for string as parameter
5) Use Preferences Manager from the exercises inside the game (could be used for player profiles and keeping high scores)
6) Implement more custom shaders
7) More options for tweaking particles (size, emission type, more variants of interpolation and more)
8) Implement sound and music using SDL_mixer

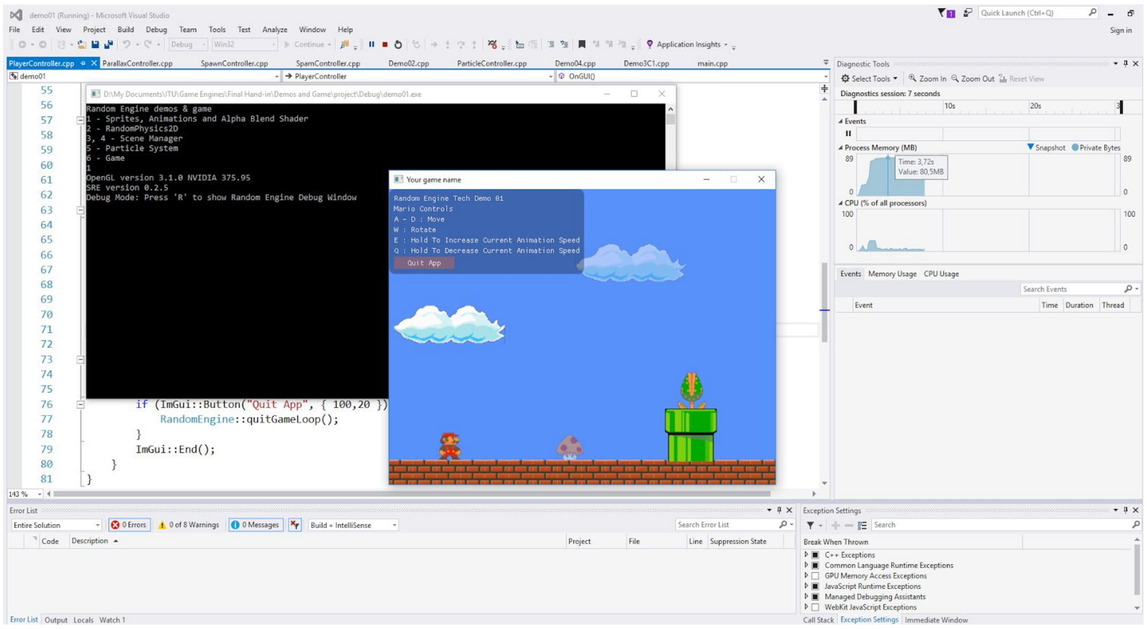
Comments

1) SpriteAtlas::addFromFile() calls SRE::Texture::createFromFile(), which creates a memory leak per use
2) We only support top level Body2DCmp, we couldn't fix the globalRotation() to update colliders and transforms properly in a tree structure
3) It is a bad idea to play in the inspector with the transform properties of a GameObject with Body2DCmp

IV. Demos and Game

Sprite - Animation - AlphaChannel Shader

Showcase about how you can import Sprites with SpriteAtlas, which parse TexturePacker json files. You can see how Animation works and the control you can have over it. You can see on display some of the sprites using our own AlphaBlend shader.


demo01 (Running) - Microsoft Visual Studio
PlayerController.cpp
D:\My Documents\ITU\Game Engines\Final Hand-in\Demos and Game\project\Debug\demo01.exe
Your game name
Diagnostic Tools
Error List
Exception Settings

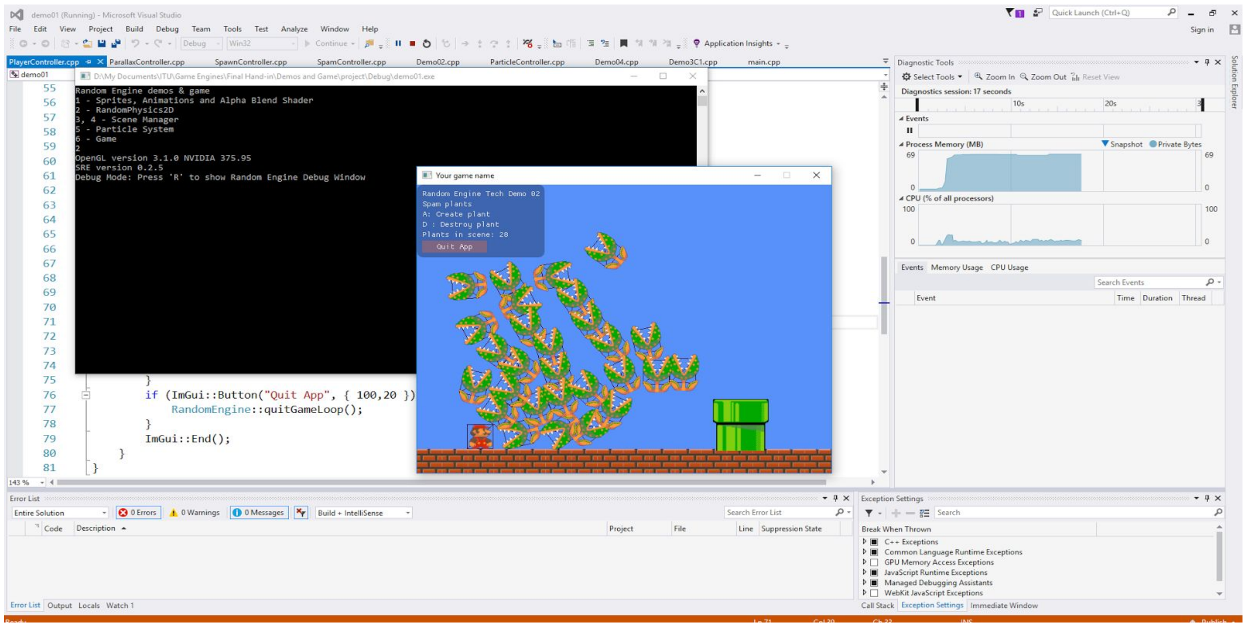
```
Random Engine demos & game
1 - Sprites, Animations and Alpha Blend Shader
2 - RandomPhysics2D
3, 4 - Scene Manager
5 - Particle System
6 - Game
1
OpenGL version 3.1.0 NVIDIA 375.95
SRE version 0.2.5
Debug Mode: Press 'R' to show Random Engine Debug Window
```

```
Random Engine Tech Demo 01
Mario Controls
A - D : Move
W : Rotate
E : Hold To Increase Current Animation Speed
Q : Hold To Decrease Current Animation Speed
Quit App
```

```
        if (ImGui::Button("Quit App", { 100,20 })
            RandomEngine::quitGameLoop();
        }
        ImGui::End();
    }
}
```

RandomPhysics2D

Showcase about how Random Engine supports dynamic bodies, and how the engine handles the destruction of game objects in runtime



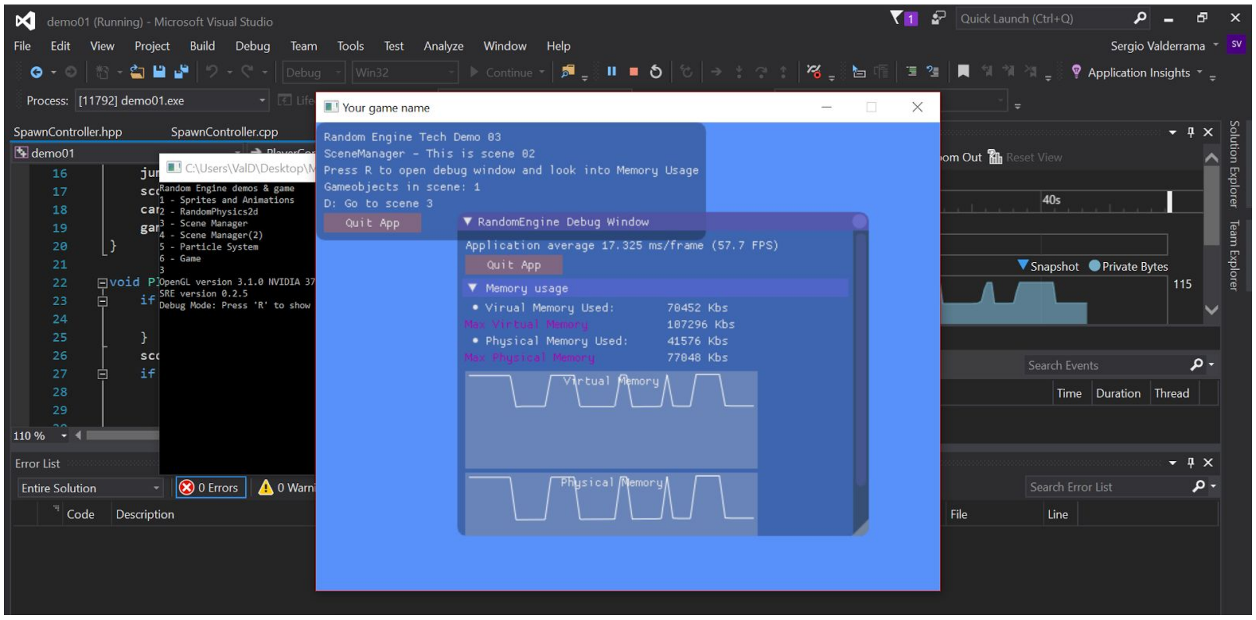
```
Random Engine demos & game
1 - Sprites, Animations and Alpha Blend Shader
2 - RandomPhysics2D
3, 4 - Scene Manager
5 - Particle System
6 - Game
2
OpenGL version 3.1.0 NVIDIA 375.95
SRE version 0.2.5
Debug Mode: Press 'R' to show Random Engine Debug Window
```

```
Random Engine Tech Demo 02
Spam plants
A: Create plant
D : Destroy plant
Plants in scene: 28
```

```
        }
        if (ImGui::Button("Quit App", { 100,20 })
            RandomEngine::quitGameLoop();
        }
        ImGui::End();
    }
}
```

Profiler and Scene Manager

Showcase how Random Engine supports scene transition, and how it successfully deletes all game objects from a scene. You can see that data with the help of our memory profiler



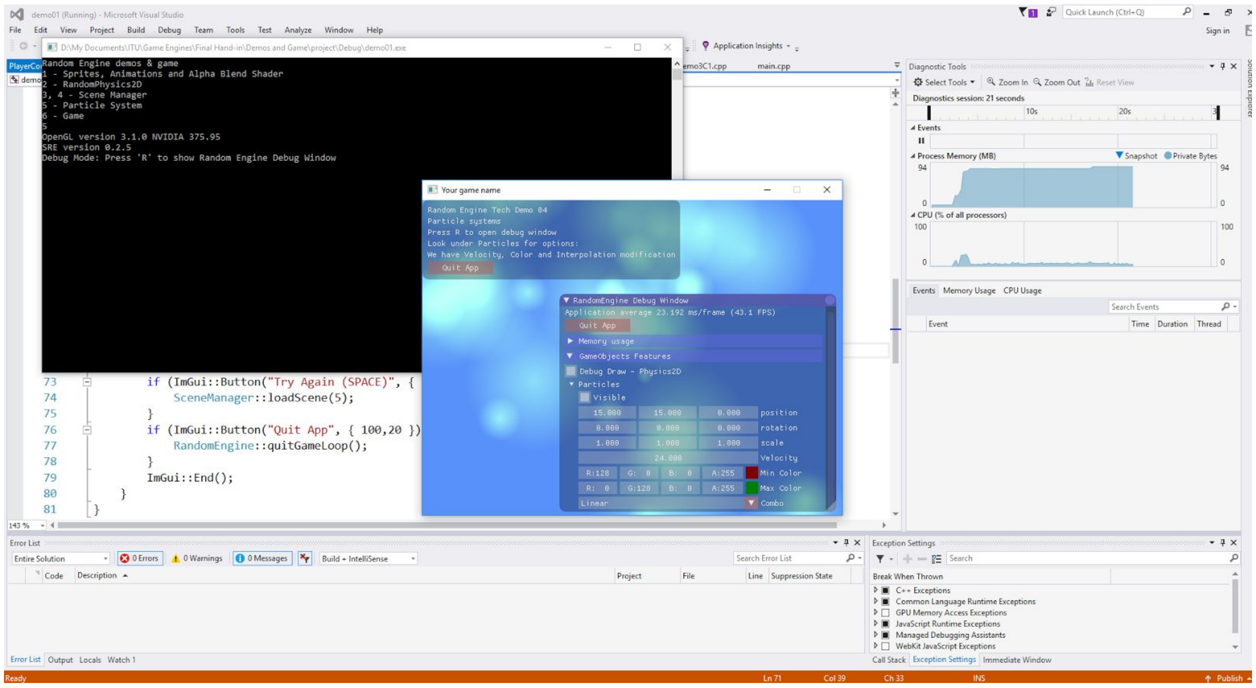
```
Random Engine demos & game
1 - Sprites and Animations
2 - RandomPhysics2d
3 - Scene Manager
4 - Scene Manager(2)
5 - Particle System
6 - Game
3
OpenGL version 3.1.0 NVIDIA 37
SRE version 0.2.5
Debug Mode: Press 'R' to show
```

```
Random Engine Tech Demo 03
SceneManager - This is scene 02
Press R to open debug window and look into Memory Usage
Gameobjects in scene: 1
D: Go to scene 3
```

```
RandomEngine Debug Window
Application average 17.325 ms/frame (57.7 FPS)
Memory usage
 • Virual Memory Used:        70452 Kbs
Max Virtual Memory            107296 Kbs
 • Physical Memory Used:      41576 Kbs
Max Physical Memory           77048 Kbs
```

Particle system

Showcase of our engine's Particle system; we can manipulate the color and velocity of particles and use Linear, Bezier and Random interpolation.



```
Random Engine demos & game
1 - Sprites, Animations and Alpha Blend Shader
2 - RandomPhysics2D
3, 4 - Scene Manager
5 - Particle System
6 - Game
5
OpenGL version 3.1.0 NVIDIA 375.95
SRE version 0.2.5
Debug Mode: Press 'R' to show Random Engine Debug Window
```

```
Random Engine Tech Demo 04
Particle systems
Press R to open debug window
Look under Particles for options:
We have Velocity, Color and Interpolation modification
```

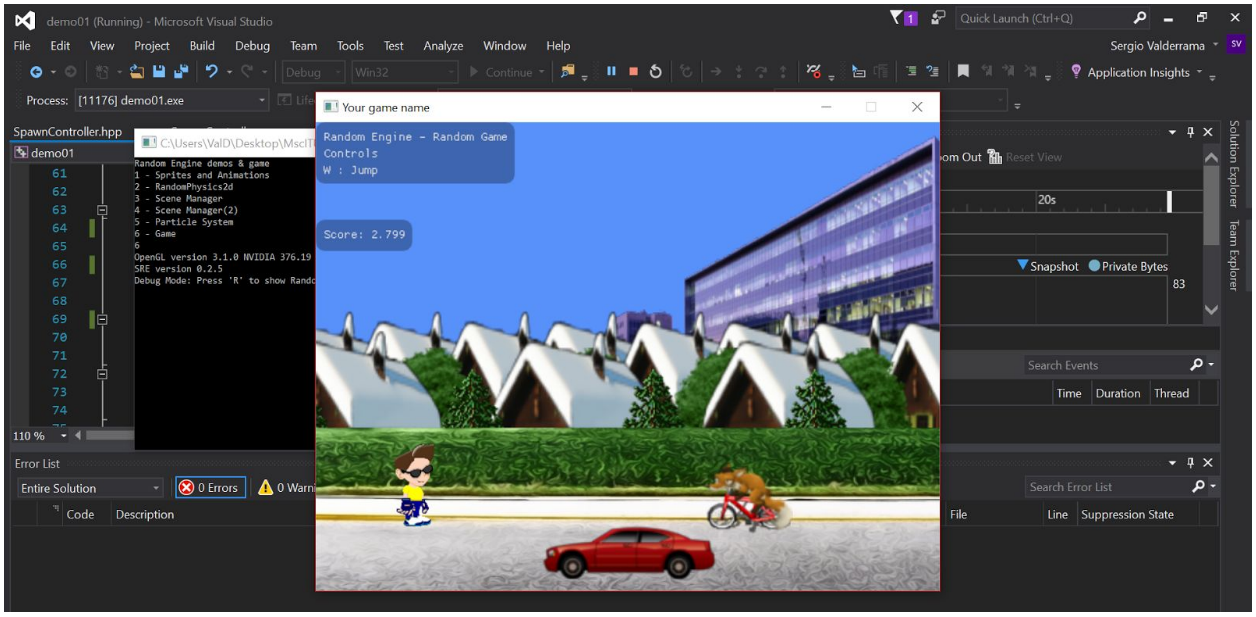
```
RandomEngine Debug Window
Application average 23.192 ms/frame (43.1 FPS)
Debug Draw - Physics2D
Particles
  Visible
  15.000   15.000   0.000   position
  0.000    0.000    0.000   rotation
  1.000    1.000    1.000   scale
  24.000                    Velocity
  R:128  G: 0    B: 0  A:255  Min Color
  R: 0   G:128   B: 0  A:255  Max Color
  Linear                      Combo
```

```
        if (ImGui::Button("Try Again (SPACE)", {
            SceneManager::loadScene(5);
        }
        if (ImGui::Button("Quit App", { 100,20 })
            RandomEngine::quitGameLoop();
        }
        ImGui::End();
    }
}
```

demo01 (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
D:\My Documents\ITU\Game Engines\Final Hand-in\Demos and Game\project\Debug\demo01.exe
Your game name
Quit App
Memory usage
GameObjects Features
Diagnostic Tools
Diagnostics session: 21 seconds
Process Memory (MB)
CPU (% of all processors)
Error List
0 Errors
0 Warnings
0 Messages
Exception Settings
Ready

Random Game

We decided to make an infinite runner game to show how to make a game with Random Engine using some of the features that we have in the engine, and we had a lot of fun making it!



```
Random Engine demos & game
1 - Sprites and Animations
2 - RandomPhysics2d
3 - Scene Manager
4 - Scene Manager(2)
5 - Particle System
6 - Game
6
OpenGL version 3.1.0 NVIDIA 376.19
SRE version 0.2.5
Debug Mode: Press 'R' to show Randc
```

```
Random Engine - Random Game
Controls
W : Jump

Score: 2.799
```

V. Engine build notes

The engine needs to be built with CMake. Included in the submission are the external libraries needed, minus SRE.
SRE needs to be built separately in CMake. We use an older version of SRE that can be found on GitHub here:
https://github.com/mortennobel/SimpleRenderEngine/tree/aff88a74128cf519439d9e7c300ff33ea3be7d7b
Once Random Engine has been built, the demos also need to be built in CMake.

VI. Responsibility table

Component\Name	Atanas	Felipe	Sergio
Animations and Sprite related cmp	X		X
Physics2D			X
Particles	X		
GUI			X
Input	X		
Materials	X		X
Scene Manager			X
Scriptable Component			X
Transform Component	X		
Profiler and Memory Leaks Detector			X
Shaders		X	
Documentation	X	X	X