|  |  |
| --- | --- |
| **GAT150 – Introduction to Game Programming** |  |

***SCENE***

***OVERVIEW***

In this assignment you will create a game scene to read, update and draw all the game objects in the game. The scene class will hold a container (list) of objects and perform various functions on the objects in the container.

***GRADING***

This assignment is worth 50 points. To receive full credit for this assignment, you’ll need to show the game running with an object contained in the scene read from a JSON file.

* 40 points – Read scene file with player sprite
* 10 points – Read scene file with player and background sprites (load array of game objects)

***INSTRUCTIONS***

# Create a Scene class

* Create a Scene.h file in the *Objects* folder in *Engine*
* Create a *Scene* class in its own namespace

**Scene.h**

namespace nc

{

class Engine;

class GameObject;

class Scene : public Object

{

public:

virtual void Create(void\* data = nullptr) override;

virtual void Destroy() override;

void Read(const rapidjson::Value& value) override;

void Update() override;

void Draw();

GameObject\* Find(const std::string& name);

void AddGameObject(GameObject\* gameObject);

void RemoveGameObject(GameObject\* gameObject);

void RemoveAllGameObjects();

protected:

Engine\* m\_engine{ nullptr };

std::list<GameObject\*> m\_gameObjects;

};

}

* Create the definitions for these in Scene.cpp (Highlight methods and press Ctrl+.)

**Scene.cpp**

#include "pch.h"

#include "Scene.h"

#include "Actor.h"

namespace nc

{

void Scene::Create(void\* data)

{

m\_engine = static\_cast<Engine\*>(data);

}

void Scene::Destroy()

{

// call the member function to remove all game objects

}

void Scene::Read(const rapidjson::Value& value)

{

const rapidjson::Value& objectValue = value["GameObject"];

if (objectValue.IsObject())

{

std::string typeName;

// read component “type” name from json (Get)

nc::GameObject\* gameObject = // get from object factory, use typeName as the key

if (gameObject)

{

gameObject->Create(m\_engine);

// call game object read (pass in objectValue)

// call AddGameObject passing in the game object

}

}

}

void Scene::Update()

{

// iterate through the actors and call Update on each actor

for (auto gameObject : m\_gameObjects)

// update

}

void Scene::Draw()

{

// iterate through the actors and call Draw on each actor

for (auto gameObject : m\_gameObjects)

// draw

}

GameObject\* Scene::Find(const std::string& name)

{

for (auto gameObject : m\_gameObjects)

{

// compare game object name to name parameter (==)

{

return gameObject;

}

}

return nullptr;

}

void Scene::AddGameObject(GameObject\* gameObject)

{

// push back gameObject

}

void Scene::RemoveGameObject(GameObject\* gameObject)

{

auto iter = std::find(*begin*, *end*, gameObject);

if (iter != *end*)

{

// destroy (\*iter)->

// delete

// erase iter from m\_gameObjects

}

}

void Scene::RemoveAllGameObjects()

{

for (GameObject\* gameObject : m\_gameObjects)

{

// destroy

// delete

}

// clear game objects

}

# Update GameObject to read components

* Add a method to read an array of components from the JSON file

**GameObject.h**

void ReadComponents(const rapidjson::Value& value);

**GameObject.cpp**

void GameObject::ReadComponents(const rapidjson::Value& value)

{

for (rapidjson::SizeType i = 0; i < value.Size(); i++)

{

const rapidjson::Value& componentValue = value[i];

if (componentValue.IsObject())

{

std::string typeName;

// read component “type” name from json (Get)

Component\* component = // create component from object factory

if (component)

{

// call component create, pass in gameobject (this)

// call component read

// add component to game object

}

}

}

}

* Update the GameObject Read function to read components in an array

void GameObject::Read(const rapidjson::Value& value)

{

json::Get(value, "name", m\_name);

json::Get(value, "position", m\_transform.position);

json::Get(value, "scale", m\_transform.scale);

json::Get(value, "angle", m\_transform.angle);

**const rapidjson::Value& componentsValue = value["Components"];**

**if (componentsValue.IsArray())**

**{**

**ReadComponents(componentsValue);**

**}**

}

# Create Scene and read JSON scene file

**Main.cpp**

nc::Scene scene;

int main(int, char\*\*)

{

scene.Create(&engine);

rapidjson::Document document;

nc::json::Load("scene.txt", document);

scene.Read(document);

...

// call scene update after engine update

...

// call scene draw after background draw

// call scene destroy before engine shutdown (outside of while loop)

# Scene file example

**Scene.txt**

{

"GameObject":{

"type":"GameObject",

"name":"Player",

"position":[

300,

400

],

"scale":1,

"angle":45,

"Components":[

{

"type":"PhysicsComponent"

},

{

"type":"PlayerComponent"

},

{

"type":"SpriteComponent",

"texture":"cars.png",

"rect":[

64,

110,

60,

112

]

}

]

}

}

*You should now be able to run the program and have the scene display the controllable car. This is worth 40 points.*

# Add multiple GameObject reads

*Add the ability to read in the background as a game object also. This requires that an array of game objects can be read. The background will only have a sprite component. This is worth 10 points.*

void Scene::Read(const rapidjson::Value& value)

{

**const rapidjson::Value& objectsValue = value["GameObjects"];**

**if (objectsValue.IsArray())**

**{**

**ReadGameObjects(objectsValue);**

**}**

}

void Scene::ReadGameObjects(const rapidjson::Value& value)

{

for (rapidjson::SizeType i = 0; i < value.Size(); i++)

{

const rapidjson::Value& objectValue = value[i];

if (objectValue.IsObject())

{

std::string typeName;

// read game object “type” name from json (Get)

GameObject\* gameObject = // create game object from object factory

if (gameObject)

{

// call game object create, pass in m\_engine

// call game object read

// add game object to scene

}

}

}

}

* The scene.txt file will need to be updated to handle reading in the player and background

*Submit the scene.h and scene.cpp file for submission.*

***REFERENCES***

Information on reading JSON values, including arrays, with rapidjson

<https://rapidjson.org/md_doc_tutorial.html#QueryValue>

<https://rapidjson.org/>

JSON formatter to test JSON file format:

[https://jsonformatter.curiousconcept.com/#](https://jsonformatter.curiousconcept.com/)

JSON: references

<https://www.json.org/json-en.html>

<https://www.w3schools.com/js/js_json_intro.asp>