

Jack Porter

Q12313882

Abstract

Reflective report on the development process of the game designed in AE1

AE2 - POst Portem Report

CGP600 Advanced Games Programming AE2

Table of Contents

[Mechanics 2](#_Toc535182591)

[Movement 2](#_Toc535182592)

[Enemies 2](#_Toc535182593)

[Graphics 2](#_Toc535182594)

[FBX files 2](#_Toc535182595)

[Animations 2](#_Toc535182596)

[Lighting 2](#_Toc535182597)

[Particles 3](#_Toc535182598)

[Design Patterns 3](#_Toc535182599)

[Component System 3](#_Toc535182600)

[Model Manager 3](#_Toc535182601)

[Texture Manager 3](#_Toc535182602)

[Time Class 4](#_Toc535182603)

[Logic 4](#_Toc535182604)

[Collision 4](#_Toc535182605)

[Sphere Collision 4](#_Toc535182606)

[Triangle to Triangle Collision 4](#_Toc535182607)

[Capsule Collision 4](#_Toc535182608)

[Capsule to Box Collision 5](#_Toc535182609)

[AI 5](#_Toc535182610)

[Gravity 5](#_Toc535182611)

[Testing 5](#_Toc535182612)

[Conclusion 5](#_Toc535182613)

# Mechanics

## Movement

The mechanics that I have managed to implement for this game is movement, jumping and player death.

The player is able to move around the level, correctly colliding with the environment using Capsule to box collision, as well as being able to collide with other entities, using Capsule to Capsule collision.

## Enemies

There is a robot that will target the player and move towards them so long as they are on the ground, and upon contact with the player, will throw them away, dealing damage.

If the player reaches 0HP, they will be killed, and the game will end.

# Graphics

## FBX files

One feature I was able to implement that I am very proud of is the ability to load in an FBX file instead of an OBJ file to the game, this required the use of the FBX SDK (REFERENCE) and was implemented with the help of two blog posts (REFERENCE), (REFERENCE) and the FBX SDK documentation (REFERENCE).

There is an issue, however, with s loaded model if that model that contains animation data, this results in some of the triangles not being rendered, and I could not find a solution to the problem.

## Animations

I also tried to use the animation data that can be stored inside an FBX file to add animations to my game, however, even with the help of (REFERENCE), which focuses on FBX files and (REFERENCE) which talks about skeletal animation in a project, I could not work out how to implement the animations into my game, however, the groundwork is there, and the animation data that is required, including bone structure, vertex offsets and frame timing, is saved in the model when it is loaded.

I did manage to add in a simple animation to my player, based on 2D sprite animation, using multiple OBJ files that get loaded in one after another in sequence, which with the use of my model manager, means that each model is only loaded in once and then stored in memory, meaning that once the animation is loaded, it is very efficient at swapping between “frames”.

## Lighting

## Particles

My particle system has 3 different effects that I have created:

A stream that emits from behind the player in the opposite direction when they move.

A fountain that can be spawned in the middle of the map by holding down the O key on the keyboard, this also runs when the player collides with an enemy, however it only spawns a small number of particles and isn’t very visible.

A snow particle system that creates a lot of particles that fall from above in the scene in 3D space, this is very resource intensive though.

All the particles that I am using are in an object pool and are created when the game starts, this means that I am not creating objects at run time every time I want to create a particle, which makes the game far more performant. Each particle also has a texture associated to them and will switch based on what particle is being used, these textures also use alpha blending so that you can see particles behind others.

# Design Patterns

## Component System

For this project I had planned on implementing a component system so that I could easily add functionality to my game objects, this turned out to be simple once I found out about type\_info, typeid, typename and template. These allows me to run functions such as GetComponent<Physics>(); and return a Physics component attached to a game object.

The components are all stored in the parent game object inside of a map, with the class or type\_info being the key, they are added to this map using their typeid and are found in functions such as GetComponent<>() using the typename. Template is what allows me to add the typename to the function call inside the angled brackets.

## Model Manager

I created a model manager in my game, it handles the creation and pooling of models, and ensures that I never have to create the same model twice. It does this by storing each model as a pointer inside a map, with the key being the filename of the model, this means that I only need to load the file name and the model manager will handle the rest.

It works by first checking the map to see if any model matching the provided key, the file name, exists, if it does it simply returns a pointer to that model, if it doesn’t exist it will first create the model, and then add it to the map under the file name key, and then return a pointer to the newly created model.

## Texture Manager

I also incorporated a texture manager to the game, it works in much the same way as the model manger, only creating and returning pointers to a texture and sampler instead. This means that each model can have a different texture even if they share the same model data, likewise two different models can use the same texture.

## Time Class

Finally I created a Time class, this is really important for my game as all my movement is based no delta time, the time in seconds between each tick, or frame drawn. I used the book “Introduction to 3D game programming with DirectX 11” (REFERENCE) to create this class.

# Logic

## Collision

### Sphere Collision

For collision I started with Sphere collision as it is the simplest way to detect if two objects are colliding, this was done by getting the vector between the two checked models positions + their centre points, and then checking if the magnitude of that vector, if the magnitude is less than the sum of their radii then a collision is happening.

Detecting the centre point in the model is done by looping through each vertex in the model and storing the maximum and minimum x, y and z positions, then finding the middle point of each of those extremes, giving me the middle point in the model.

Working out the radius is done by getting a vector between the centre point of the model and each vertex, the largest magnitude of all the vectors will be set to the radius as this is the smallest size that will cover every vertex in the model.

### Triangle to Triangle Collision

### Capsule Collision

After I had failed to implement Triangle to Triangle Collision I decided to go with a slightly easier solution of capsule collision, this is preferable as my player model is of a humanoid shape and so a sphere will give a very unrealistic bounding area for where the player should be colliding, but a capsule will give a much smaller radius area whilst allowing me to check the entire height of the player model.

Capsule collision requires having a height value and a radius value:

To get the height value I just take the distance between the minimum Y and maximum Y values when looping through all the vertices and then subtract the radius twice from that distance, this is because the radius will be added to the point in that height that I am checking at so I do not want to go out of the bounds of the model in the Y axis.

To get the radius I check the largest distance in the z direction for all vectors and set that, this is a bit of a cheating solution as I know that my model will default to a T-Pose and so checking on the X and Z positions will yield the radius to be the distance to the end of the arms.

To check capsule collision, you must first find the closest Y point on each model that you can check against, once you have that y position on each model, you do a normal sphere collision check with the new y position subbed in instead.

### Capsule to Box Collision

Capsule to box collision is relatively simple once you understand how capsule collision works, the only difference is that instead of working out the Y position of the box to check against, you still work this out for the capsule, you instead work out the closest point on each box axis to the player, if they are outside the bounds of the box, then you take the extreme, if they are inside the boxes bounds then you take the players position for that axis.

I worked out the bounds of the box much the same way as I worked out the centre point of the model, only instead of finding the middle point of each extreme, I saved them as a variable.

## AI

The AI that I programmed for this game is incredibly simple, in that it looks towards the player and moves forward at a set speed, if I had more time I would have liked to have done some kind of path finding for the AI, as well as some more logic outside of just chasing the player, such as patrols.

## Gravity

For gravity I merely had a float variable set to a negative number, and on each frame I would add this float, multiplied by the delta time of the Time class, to the Y velocity of the object, this would give all objects acceleration towards the ground, to make this a little bit more controllable I also added a weight value that the gravity value is multiplied by, this means that a heavier object will fall faster than a lighter one.

In order to stop objects just falling through the floor after a set period of time, due to an increasing build-up of negative velocity until it is enough to skip past the bounds of the object it is being checked against, I had to set the y velocity to 0 each time it collided with an object, unfortunately this does mean that the objects have a tendency to get stuck against a wall if they jump into it.

# Testing

Unfortunately, due to the time constraints that I faced, I was not able to follow testing plan that was developed during the first assignment, I focused heavily on ad hoc testing during development, to see if the function that I had just programmed was not throwing and error and was working as I intended.

# Conclusion