Collision detection in video games: How to best optimise the interactions between components in a collision system?

COMP130 - Software Engineering

1706966

March 22, 2018

This paper looks into some of the different ways a collision detection system designed for use in video games, can interact with different components of the programme and best practice for setting up these interactions. Insight into how this can affect the end user will be given, then the paper will take a look at how some of the industry standard development kits handle collision detection system cohesion. Afterwards looking at design patterns that can be used to give a base when implementing a system that isn't pre built. Going on to summarise what practices should be implemented to get the best optimisation between the collision system and other components.

1 Introduction

Forms of collision detection systems have played a part in video games since the beginning, being used to simulate the world which they are creating. The basic premise of a collision system is to detect when two objects come into contact with each other in the game world, the collision system will then send information to other components informing them of the collision with many possibilities for additional data to be sent, such as location of collision, what the objects are etc. For example in a game of pong, as the ball hits a paddle, this is detected and its information is gathered and sent to another component that changes how the ball now moves based on the information given. This is a very rudimentary example, most games now have much more complex systems, making it more important to optimise how systems interact. Having only the data that is needed sent to the other component saves on memory and time.

If a system isn't working as intended or is poorly designed it can cause dissatisfaction for the end user. Things might occur such as a character falling through the floor, or being stuck on something that they can't see because the detection isn't fine enough. Take "Big Rigs: Over the Road Racing"[1] as an example, the collision for bridges in this game didn't work as would be expected, so when driving over the player's truck would fall through the bridge. There are other reasons a bad collision system could cause dissatisfaction that might not be as obvious that it's caused by the collision system. If a system is poorly optimised it can slow down the game causing lag issues and poor frames per second.

Some game engines come with ways of handling collision detection built into them, such as unity and unreal[2], we will be taking a look at how these work. When writing a programme without using a game engine there are a few different options available for implementing a collision detection system and how it interacts with components. Some of which being design patterns that can be taken as a base and worked into your code to fit your needs[3].

There are also some open source collision detection libraries that can be downloaded and used[4].

2 Game Engines

2.1 Unity

Within Unity you can give objects either colliders or rigidbodies (The colliders and rigidbodies have to be set on the object and given measurements.) both of these give the object the ability to call "OnCollisionEnter" when they collide with other objects that have one of the aforementioned properties. A collision class is passed when OnCollisionEnter is used, the class contains information about contact points impact velocity and more. If you are not planning to use this information you can leave out a parameter to save unnecessary calculation[5]. Alternatively you are able to use "OnTriggerEnter" which does a similar thing but is called when a collider enters a set trigger area. Only the fact the event has been triggered is sent as data, and that information is sent only to the collider that hit the trigger and the trigger itself[6]. Therefore a lot less data is passed in comparison to OnCollisionEnter, and its being sent to very specific locations. Unity has supplied us with some very easy to implement ways of checking for collisions, but it does have its caveats. Both objects need to be either colliders or rigidbodies. One of them must be a rigidbody in the case of the trigger event. Data that these events supply is also restricted, having basically an all or nothing approach.

2.2 Unreal

In Unreal Engine the set-up of how the colliders interact with each other is similar to unity, all objects you want to have collision need to be given a static mesh first. A static mesh can be wrapped around an object as finely as required. A more precise wrap will require more resources[7]. Once an object has been given a static mesh and been set to collision, you are then able to designate the type of collision. There are many different types along with custom grouping systems, but a few main ones will be listed[8]. First is blocking, both objects block the other from passing, but no event is generated from the collision. This is used for when you want to stop an object but don't need to know anything about it. Next we have Hit Events, similar to blocking neither object can pass through the other but this time we have an event activate. An event can be triggered on either object or both, and the event will tell any code attached to it that the object has hit another object. Information about the other object can be passed to other components. Lastly there is Overlap Events, these activate when the two objects overlap one another, so they don't block like before but instead pass through. The objects can either ignore this or send data similar to the Hit Event data[9]. Unreal seems to show a more easy to use customisation when it comes to setting up collision on objects and how they send data to other area's. However if you want to inform an component that isn't one of those colliding the programmer would need to code that inside one of the objects.

3 Design Patterns

"In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design." [3] these patterns are often used to speed up development but also to prevent issues that can cause problems later in the project. When is comes to working with collisions one specific design pattern stands out, that's the Visitor Pattern[10]. The visitor pattern allows the programme to traverse a data structure and operate on each element in the structure. A disadvantage to this is that for every new class type in the data structure a new Visit function must be added to the visitor interface and a new Accept function to the class itself[11]. The visit functions are called by the visitor for every element in the data structure and the accept function called on the class being visited. With a visitor pattern implemented there exists the ability to check through an array of all the in game objects and call a check there to see if they are colliding with another object. Using this pattern allows you to check for collision but you are not able to control the order that the visitor iterates through the data[11].

4 Libraries

With programmes having the ability to incorporate libraries smoothly into them it opens a whole world of different possibilities[12]. With this it's possible to include a library specific to or containing pre built code used for collision detection. One such library is FCL[13], although this one was designed with physical robots in mind, it can also be used for virtual interactions as well.

5 Conclusion

To conclude, given what the research in this paper has shown it is possible to come away with a few good programming practices that will help towards having a well optimised collision system.

1. Only send information where you need it if possible - Avoiding sending extra information saves on resources.

2. Limit that information to the minimum required for the component receiving the data -In a similar manner this is another step to lowering to

amount of data being passed.

3. Code the detection system be more customisable - this is the key to letting you be more specific in what information is gathered on collision, in turn letting you dispatch that information more accurately.

References

[1] 2018, accesed on 2018-03-22. [Online]. Available: https://en.wikipedia.org/wiki/Big_Rigs:_Over_the_Road_Racing

[2] P. E. Dickson, J. E. Block, G. N. Echevarria, and K. C. Keenan, "An experience-based comparison of unity and unreal for a stand-alone 3d game development course," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education.* ACM, 2017, pp. 70–75.

[3] 2018, accesed on 2018-03-22. [Online]. Available: https://sourcemaking.com/design_patterns

[4] 2018, accesed on 2018-03-22. [Online]. Available: https://github.com/jslee02/awesome-collision-detection#libraries

[5] U. Technologies, "Unity - scripting api: Collider.oncollisionenter(collision)," 2018, accesed on 2018-03-18. [Online]. Available: https://docs.unity3d.com/ScriptReference/Collider.OnCollisionEnter.html

[6] ——, "Unity - scripting api: Collider.ontriggerenter(collider)," 2018, accesed on 2018-03-18. [Online]. Available: https://docs.unity3d.com/ScriptReference/Collider.OnTriggerEnter.html

[7] 2018, accesed on 2018-03-22. [Online]. Available: https://docs.unrealengine.com/en-us/Engine/Content/Types/StaticMeshes

[8] 2018, accesed on 2018-03-22. [Online]. Available: https://www.unrealengine.com/en-US/blog/collision-filtering

[9] E. Games, “Collision overview,” 2018, accesed on 2018-03-18. [Online]. Available: https://docs.unrealengine.com/en-US/Engine/Physics/Collision/Overview

[10] E. Gamma, *Design patterns: elements of reusable object-oriented software.* Pearson Education India, 1995.

[11] B. Horsfall, N. Charlton, and B. Reus, “Verifying the reflective visitor pattern,” in *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs.* ACM, 2012, pp. 27–34.

[12] R. G. Kula, D. M. German, T. Ishio, A. Ouni, and K. Inoue, “An exploratory study on library aging by monitoring client usage in a software ecosystem,” in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on.* IEEE, 2017, pp. 407–411.

[13] J. Pan, S. Chitta, and D. Manocha, “Fcl: A general purpose library for collision and proximity queries,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* IEEE, 2012, pp. 3859–3866.