What are the difficulties of implementing algorthims that proceduraly generate game maps, in a highly coupled system?

COMP160- Software Engineering

1607804

March 28, 2017

1 Introduction

The implementation of new functionality to any existing system comes with complications. When working in the game industry one may be called upon to work on a fully developed system. In these cases, the pre-existing structure of the system may be poor. In the following essay, the author aims to provide insight on the difficulties of adding new procedural content generation (PCG) algorithms. Specifically for the generation of game maps in systems that contain classes that are tightly coupled. We define game maps as the physical environment the player has to transverse whilst playing the game. The author aims to provide a good description of common types of coupling that may arise from high interdependence. This may allow the reader to notice if any of these types are apparent in any system they are working on. By extrapolating research theory and case studies from the software engineering industry, the author will suggest

stratergies to avoid pitfalls when implementing PCG algorithms to systems which have high interdependence between its classes.

2 Discussion

Level design can be a tedious cycle of designing and playtesting to achieve the perfect level. As the design of the level should aim to be challenging enough for the player to enjoy, but not too challenging it discourages the player [1]. This process is not only time-consuming for the game designer(s) but also expensive for the company employing the game designer(s). One way to achieve a well-designed level is by automating this process through the use of PCG. Not only will this save the resources spent in the design of the map but also may keep the players engaged for longer if done correctly.

When choosing the objects that will make up the generated map, consider carefully if any of them contain traces of content coupling. Content coupling "is considered the worst type of coupling" [2, p. 95]. This occurs when objects rely on instanced variables or methods in another object to be functional. To determine whether a system is highly coupled or not, there are various different metrics used. Which metric to use is dependent on the type of type of coupling that needs to be measured. One way to measure content coupling is through the metric called Coupling between objects(CBO)[3]. The CBO of a class is equal to the number of other classes that it is coupled to.

If the CBO values for a class is high "the reusability of a class will decrease" [4, p. 468]. This means that not only is the code difficult to understand but also the one can not be changed independently of the other. When this is applied to map generation the results may be unpredictable. This high interdependence typically only occurs in dynamic objects, objects that need to respond to others and alter their behaviour on the behaviour of other world objects. By limiting yourself to using only static objects in the generation algorithm one can avoid behavioural uncertainty. Using a set of building blocks manually designed rooms to populate your map will result in a map that will look

different with each generation but will fundamentally be the same. One can observe this approach in the games like diablo[5].

Systems which contain classes that have high CBO values are at risk having circular dependencies between these classes. Circular dependencies occur when two or more modules are dependent on each other to function correctly[6]. By trying to implement a class or function with circular dependence the end result could be a deadlock or a loss of data [6]. When implementing an algorithm that uses modules that have such dependencies special care must be taken to call them in a way so that they will be successfully completed. To identify which parts of your system that may contain circular dependencies consider using static type checking systems[7, 8]. These systems can be used to detect deadlocks at compile time, however "it cannot identify all deadlocks in weakly typed languages such as C or C++"[9, p.76].

Another type of coupling often seen in poorly designed game systems is common coupling. Common coupling is where methods are coupled by the data they accsess[2]. For example, two methods that access and change the same variable in a shared, global data space would be demonstrating common coupling. Theses types of variables are known as global variables. Some older object-orientated programming languages do not give you the ability to use global variables. Working with a system that has classes using global variables one has to be careful how these classes are using the variables. If the class is editing the global variables all other classes that also use the same global variable will be altered as well. This could cause integer overflow, integer overflow happens at run time when the integer value stored exceeds the maximum value its type can store[10]. If game maps are generated with these errors behavioural uncertainty and other bugs are common. A solution could be to try and reduce the scope of the variables used. Also changing the way the variables are used in the object could reduce common coupling. For example, copying the value rather than using the actual value of the variable. In the field it is accepted that global variables are usually harmful to a system[11] however

other more recent work conflicts with this idea. Some believe that when "Used with discipline, global variables are useful"[12, p.338] .

3 Conclusion

This essay highlighted some of the difficulties of implementing PCG algorithms that are responsible for the generartion of maps. We focused on circular dependencies, common coupling and content coupling. Giving a simple to use metric so that it will help to identify levels of coupling in a system. The author hopes this work will create awareness about good practices when it comes to code design but also to allow those working in an exsisting system better predict any difficulties they may find.

References

[1] F. Kayali and J. Schuh, "Retro evolved: Level design practice exemplified by the contemporary retro game," in *Proceedings of DiGRA 2011 Conference: Think Design Play*, 2011.

[2] L. C. Briand, J. W. Daly, and J. K. Wust, "A unified framework for coupling measurement in object-oriented systems," *IEEE Transactions on software Engineering*, vol. 25, no. 1, pp. 91–121, 1999.

[3] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[4] N. P. S. Rathore and R. Gupta, "A novel coupling metrics measure difference between inheritance and interface to find better oop paradigm using c," in *Information and Communication Technologies (WICT), 2011 World Congress on.* IEEE, 2011, pp. 467–472.

[5] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, "Procedural content generation: Goals, challenges and actionable steps," in *Dagstuhl Follow-Ups*, vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[6] S. Nair and R. Jetley, "Solving circular dependencies in industrial automation programs," in *Industrial Informatics (INDIN), 2016 IEEE 14th International Conference on*. IEEE, 2016, pp. 397–404.

[7] C. Boyapati, R. Lee, and M. Rinard, "Ownership types for safe programming: Preventing data races and deadlocks," in *ACM Sigplan Notices*, vol. 37, no. 11. ACM, 2002, pp. 211–230.

[8] M. Naik, C.-S. Park, K. Sen, and D. Gay, "Effective static deadlock detection," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*. IEEE, 2009, pp. 386–396.

[9] H. K. Pyla and S. Varadarajan, "Avoiding deadlock avoidance," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*. ACM, 2010, pp. 75–86.

[10] P. Muntean, M. Rahman, A. Ibing, and C. Eckert, "Smt-constrained symbolic execution engine for integer overflow detection in c code," in *Information Security for South Africa (ISSA), 2015*. IEEE, 2015, pp. 1–8.

[11] W. Wulf and M. Shaw, "Global variable considered harmful," *ACM Sigplan notices*, vol. 8, no. 2, pp. 28–34, 1973.

[12] S. McConnell, *Code complete*. Pearson Education, 2004.