How game code and logic can be made more portable

COMP160 - Software Engineering Essay

1603748

March 19, 2017

abstract

1 Introduction

In this essay I plan to talk about how game logic and code can be more easily ported to other platforms and engines which will result in more cost effective portability of games to multiple different platforms. I will discuss the benefits of portable logic as well as the ways logic can be made portable between different game engines. I will discuss the feasibility of separating out the logic by representing it using ontologies and rules, and by introducing middleware (an events space) between the logic and the game engine [1]. I will also highlight the advantages and disadvantages of using blueprints to determine whether they are still worth using.

2 Benefits of portable logic

The logic of any game is the core of the game, it is what determines pretty much everything about the game other than aesthetics. Game engines require the logic to be its own format, for example when making games in the unreal engine it requires you to use blueprints or format your C++ code in the unreal format which can only be executed in the unreal engine. If the logic was separate from the rest of the system it is possible to port the logic to multiple game engines. This would increase logic re-usability amongst projects, as a person could migrate it to a familiar engine and thus avoid the time required learning a new engine [1]. It would also increase the scalability possibilities for the logic, depending on the future development of game engine capabilities [1].

3 How logic can be ported

It is possible to use the same logic on different game engines by using an events space [1]. The events space is used to separate the logic from the engine. This method of porting logic has been demonstrated using the Torque engine and a bespoke simulation engine. Logic for an accident scenario created as part of a knowledge-gathering exercise for police was serviced to both game engines, thus demonstrating logic portability. The STB (set-top box) game architecture design [2] uses a similar method to port STB games to different STB environments. The game architecture is divided into three subsystems: the game space, the adapters and the STB environment. Just like the events space, the game space contains the game logic and game state which is separate from the rest of the system. “The effect this has on portability is that when migrating to a new STB environment, the elements in the game space (i.e., the game state, object model, and game logic) can stay intact"[2].

4 Conclusion

References

[1] A. BinSubaih, S. Maddock, and D. Romano, "Game logic portability," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ser. ACE '05. New York, NY, USA: ACM, 2005, pp. 458–461. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/1178477.1178580

[2] J. Huang and G. Chen, "Digtal stb game portability based on mvc pattern," in *2010 Second World Congress on Software Engineering*, vol. 2, Dec 2010, pp. 13–16.