palgrave pivot

The Unity Game Engine and the Circuits of Cultural Software

Benjamin Nicoll
Brendan Keogh

palgrave
macmillan

The Unity Game Engine and the Circuits of Cultural Software

Benjamin Nicoll · Brendan Keogh

The Unity Game Engine and the Circuits of Cultural Software

palgrave
macmillan

Benjamin Nicoll
Digital Media Research Centre,
School of Communication
Queensland University of Technology
Brisbane, QLD, Australia

Brendan Keogh
Digital Media Research Centre,
School of Communication
Queensland University of Technology
Brisbane, QLD, Australia

ISBN 978-3-030-25011-9 ISBN 978-3-030-25012-6 (eBook)
https://doi.org/10.1007/978-3-030-25012-6

Cover illustration: © John Rawsterne/patternhead.com
Cover design by eStudioCalamar

This Palgrave Pivot imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Acknowledgements

This book represents a synthesis of two separate research projects, managed independently by both authors. Benjamin Nicoll thanks the University of Melbourne's Networked Society Institute, Intellectual Property Research Institute of Australia, and Centre for Media and Communications Law for jointly funding two collaborative research grants on game engines, which supported his research for this book in 2018. He would also like to thank Megan Richardson, Bjorn Nansen, Adam Lodders, and Jeannie Paterson for helping to secure this financial support, and for providing indispensable mentorship and opportunities for collaboration; Dan Golding, Dale Leorke, and Helen Stuckey for their collegial contributions and suggestions with regard to the above grants; John Sietsma and Chris Murphy for offering crucial insights into the world of commercial game engines; and Ceri Hutton, who was generous in accommodating his research residency at The Arcade. Benjamin also thanks his interview participants for taking the time to be interviewed, for agreeing to be quoted in publications based on the research, and for their willingness to share their thoughts and opinions on game engines. Beyond these acknowledgements, Benjamin thanks his partner, Britt, for a contribution beyond measure.

Brendan Keogh's contribution to this book was supported by the Australian Research Council Discovery Early Career Fellowship, 'Australian Game Developers and Skills Transfer' (DE180100973), and he wishes to thank all his research participants for their insights, time, and candour. He is also grateful to his friends Dan Golding,

Ben Abraham, and Terry Burdak for their ongoing support. Finally, Brendan thanks his partner, Helen Berents, not only for her endless support but also in this case for her expert feedback and suggestions on how we depict the circuits of our cultural software framework and, as an International Relations scholar, on our use of the concepts of 'governance' and 'democracy'.

Both authors would like to thank Mala Sanghera-Warren and Lucy Batrouney, our editors at Palgrave Macmillan, for their guidance and support throughout the proposal, peer review, and publication process. We would also like to thank the anonymous reviewers for their incredibly thoughtful, detailed, and encouraging comments on the manuscript.

The authors received permission to publish Fig. 1.1 from Grace Bruxner; Fig. 3.1 from Unity Technologies; and Fig. 6.1 from Bennett Foddy.

Contents

List of Figures

CHAPTER 1

The Unity Game Engine and the Circuits of Cultural Software

Abstract This chapter describes the 'circuits of cultural software' as a framework that guides the book and its analysis; offers a preliminary definition of game engines; and introduces the Unity game engine as the book's core case study. It also discusses key terms such as cultural software, proprietary and commercial game engines, workflow, grain, literacy, and governance, and situates the book in relation to existing research on videogame production, game engines, and software culture. It briefly discusses Unity's place in Australia's videogame industry—which is where the research for the book was conducted—and provides a chapter outline.

Keywords Cultural software · Unity game engine · Circuit of culture · Game engine · Software studies · Platform studies

The videogame *Grace Bruxner Presents: The Haunted Island, a Frog Detective Game* (Bruxner, 2018) is notable in its simplicity. It is approximately one hour long, and is premised on exploration, observation, and reading rather than complex systems, challenges, and goals (see Fig. 1.1). Its charming visual style and clever writing have seen it nominated for a number of awards at international videogame festivals, and it has received extensive coverage in the videogame press. Yet, *The Haunted Island* was not made in a typical videogame development environment—that is, in a studio comprised of large groups of specialist creative workers and

B. Nicoll and B. Keogh, *The Unity Game Engine and the Circuits of Cultural Software*, https://doi.org/10.1007/978-3-030-25012-6_1

Fig. 1.1 The detective inspects a bowl of pasta in *Grace Bruxner Presents: The Haunted Island, a Frog Detective Game* (Bruxner, 2018). By permission of Grace Bruxner

corporate resources. It was developed primarily by one person, Grace Bruxner, with programming and audio support from Tom Bowker and Dan Golding, respectively. Grace wrote the dialogue, modelled and animated the characters, designed the layout of the virtual world, and put together the videogame's events. Notably, Grace was able to make *The Haunted Island* while still completing a videogame design undergraduate degree at RMIT University in Melbourne. To do this, Grace took advantage of a commercial software tool known as Unity, owned by Unity Technologies.[1] Without paying any fees upfront, and without the need for low-level computer science skills, Grace used Unity to put together *The Haunted Island*'s necessary elements and export 'builds' for Windows and Mac.

Today's videogame-making ecology is increasingly inhabited by creators who, like Grace, are taking advantage of low-cost and low barrier to

[1]Where feasible, throughout this book we use 'Unity Technologies' to refer to the company and 'Unity' to refer to the game engine owned by that company. However, in popular vernacular, and thus in many of our respondents' quotes, Unity Technologies is often referred to as 'Unity'.

entry software tools to produce a wide range of videogame works. Many of these creators are no longer confined to traditional studio environments, and are instead working across a spectrum of formal and informal contexts. Several cultural and technical factors have afforded this diffusion (see Keogh 2019), but in this book we are centrally concerned with Unity. Unity is a software development tool commonly identified as a 'game engine'. Games engines enable programmers, designers, and artists to build, collaborate on, and run real-time interactive digital content, including (but not limited to) videogames. In videogame development, game engines function as software hubs wherein a vast range of media forms and skills converge into singular videogame builds. Game engines have been foundational to videogame development since at least the mid-1990s, yet the last decade has seen radical shifts in the availability and accessibility of various game engines, each with their own affordances. This, in turn, has created fertile ground for a plurality of videogame styles, genres, and developer identities to emerge, in a manner not dissimilar to the introduction of the Kodak camera or the 8-track tape. Unity holds a notable position in these shifts. Its low-cost availability, relative ease of use, and ability to scale to a vast range of student, amateur, professional, and industrial applications have seen it come to dominate videogame production globally, to such an extent that the CEO of Unity Technologies, John Riccitiello, boasts that over half of all videogame and virtual reality projects on contemporary devices are developed in Unity (Dillet 2018).

Game engines are typically owned and distributed by commercial companies that are directly invested in ensuring their engines capture a large market share. Unity, with its accessible editing interface, flexible licensing structure, and modular toolset, is framed by company representatives as an almost revolutionary piece of software that is 'democratizing game development' and 'empowering game developers' (see Unity 2018). To this end, Unity is associated with a levelling out of work role hierarchies in studio environments—hierarchies that, historically, have delegated power to programmers and software engineers as opposed to artists and designers such as Grace. Yet, while Unity claims to have democratized the means of videogame production, it has also provoked the ire (and, in some cases, outright hatred) of a small—yet vocal—group of developers, critics, and players. A brief search on any videogame enthusiast discussion board yields accusations that Unity's accessibility is causing an oversaturation of low-quality videogames,

a dearth of programming skills, and a proliferation of 'asset flipping' in videogame development—a derogatory expression referring to videogames constructed from prefabricated (i.e. store-bought) parts or assets (Grayson 2018). In a similar vein, some developers perceive a looming 'indiepocalypse' of supply overwhelming demand as a repeat of the North American videogame industry crash of 1983, which almost destroyed Atari and a national industry (Pedercini 2017). Some industry professionals and educators express concern that junior developers and students are not 'really' learning how to make videogames, but simply learning how to use Unity. Digital marketplaces, such as Valve's Steam platform, have made public promises to crack down on ostensibly 'fake games' made in Unity. Scholars, too, express concern that game engines have, since their introduction in the 1990s, led to a homogenization and rationalization of videogame production (Kirkpatrick 2013: 105–106; see also Freedman 2018a: n.p.). Game engines can also be understood in terms of a broader 'platformization of cultural production' (Nieborg and Poell 2018), wherein cultural production is increasingly controlled by a small number of dominant platform companies. These varied anxieties point to a radical reconfiguration of the practices, identities, values, and contexts associated with videogame development today.

How, then, might we make sense of these cultural, technological, and design shifts that, at once, seem to empower developers such as Grace, yet that also seem to make developers beholden to a single company's product? It is this duality that this book is centrally concerned with. In the chapters that follow, we argue that game engines are a form of *cultural software*, and that their social, political, technological, and ideological effects must be mapped and analysed. While Lev Manovich (2013: 21) defines cultural software as 'software that support actions we normally associate with culture', we adopt a narrower definition: *cultural software are software that provide code frameworks for actions we normally associate with cultural production*. We are thinking, here, of software tools such as Photoshop, Blender, Garage Band, Final Cut Pro, and, of course, Unity. Such programs, we argue, enrol their users in *circuits of cultural software* in the way they influence, mediate, and articulate the processes and contexts of cultural production. Cultural software have a fundamental bearing on production *workflows* across different design contexts. They encourage media creatives to adopt particular design methodologies and thus possess varying *grains*—protocols, standards, and affordances—that give shape to creative