Editor’s Note: In IJGCMS 2(1) and 2(2), the introductory editorials focused on preparing students to succeed in gaming and simulation careers, both within and outside of academia. In both editorials, I invited members from academia, government, and business to respond. Responses came from graduate faculty, business leaders, and graduate students. This is the first of those responses, continuing a discussion on preparing students for success. Readers interested in adding their own editorial comments should contact Dr. Rick Ferdig, IJGCMS Editor, at ijgcms@gmail.com

In addition to Habgood’s insightful comments, there are five excellent research articles in this issue of IJGCMS. Tan and her colleagues explored the role of sound while subjects played the Legend of Zelda. They found that subjects did best when listening to sound unrelated to the gameplay, raising questions about the impact of multisensory cues. Annetta and his collaborators examined a project where teachers and students create serious games. Using a mixed-method design, the researchers found further evidence of the importance of student design—a finding that led them to the creation of a new development environment. Garrett and McMahon discuss a conceptual framework they call Simulation, User, and Problem-based Learning (SUPL). They offer a design that includes a 3D simulation technology approach in order to guide learning and ensure effective training. In the fourth research paper, Heider and Massanari explore friendship, closeness, and disclosure in Second Life. The authors spent four years collecting data and found that people developed relationships in ways they might not in face-to-face settings. Wodehouse and Ion conclude the research section with a discussion of gaming scenarios for product development teams. They suggest that cues from games can help integrate information and individuals more effectively. This issue of IJGCMS ends with a book review on Gee’s Good Video Games & Good Learning.
INTRODUCTION

Academics have been highlighting the synergies between the principles of good game design and learning theory since the early 1980’s. There appears to be a level at which the communities of practice that engage in professional game development have evolved an implicit understanding of what makes good learning. Of course, no one is suggesting that Mario’s game designers knowingly base their game progression around constructivist principles, or explicitly include helpful supporting characters in recognition of the literature on scaffolding. Good learning may be part of the culture of game design, but the separate fields have yet to acquire a shared vocabulary for describing it. However, it would certainly be interesting to see whether Shigeru Miyamoto (Mario’s distinguished designer) would be able to transfer his apparent understanding of the principles of good learning to teaching in the classroom.

My own professional career in the games industry pales in comparison to Mr. Miyamoto’s, but ten years developing console games makes me a product of a similar cultural environment. My perspective is derived from a common set of attitudes, assumptions and prejudices, which make it easier to see academia through the eyes of the games industry than the other way around. Nonetheless, in the past year I have returned to the academic world and have been trying to apply my own understanding of learning theory to the practise of teaching game development at Sheffield Hallam University. I have the advantage of a PhD in Learning Sciences and a passion for the relationship between games and learning, but I am essentially a game developer trying to make an impact in the classroom. This editorial is a summary of my reflections of university teaching in light of my experience both as an employer and now teacher of game programmers.

EMPLOYING A GRADUATE

In my last position in the games industry I ran a Serious Games division within a medium-sized game development studio. The studio, as a whole, developed games for all the major consoles, and had a range of well-known clients. In my role I was involved in sifting and interviewing a large number of new graduates, and then working with the successful few on some of my projects. The games industry has always been more likely to judge a candidate on their portfolio than their degree course or classification. This may seem like a snub to academic courses, but it is really just recognition of the intensely applied nature of the occupation, and candidates would certainly be expected to have a degree as well. Nonetheless, it is the ability of a candidate to apply their academic knowledge to creating software that interests studios.

PORTFOLIOS

Traditionally, an ideal portfolio demo for a graduate seeking a position as a junior game programmer would include both a complete game and a technology demo (graphics rendering techniques, artificial intelligence and physics demos are always popular choices). A portfolio game doesn’t need to be complicated, and could even be two-dimensional. A simple game that is well polished creates a much better impression than another unfinished Grand Theft Auto clone. Nonetheless all demos should be written in C++ and ideally include the source code which compiles under Visual Studio, and runs without the need for additional library installs.

While few candidates would score full marks against these criteria, it’s usually not too hard to find competent technology demos. Nonetheless you have to be careful as the same demos can appear on the portfolios of all stu-
dents graduating from a particular university, suggesting that it is actually a tutorial aspect of their course.

Unfortunately competent examples of complete games written in C++ are much rarer. Impressive-looking Unreal 'mods' sometimes appear in portfolios, but these are unlikely to be viewed favourably by studios that aren’t using the Unreal engine. In fact most studios won’t be able to play these as they won’t have the Unreal game readily available. It’s also almost impossible for the uninitiated to distinguish the student’s programming skills from those of the talented coders at Epic. Programming in UnrealScript may be an impressive skill for a game designer to have, but a prospective game programmer should have the potential to create a scripting language in C++ and not just use one that someone else has created.

THE INTERVIEW

Interview practices in the games industry have traditionally varied widely from company to company. Nonetheless a programmer interview often includes a range of problems relating to the finer points of the C++ language, 3D mathematics, or any other specialist area of technology that the candidate has mentioned on their CV. Graduate interviewees often perform very badly in this kind of test, although it is not an attempt to catch them out or make them prove an encyclopaedic knowledge of game development. It is simply an opportunity to show that they understand the domain well enough to think and talk intelligently around relevant problems. In my experience, the graduate that is given the junior programming position is the one that has a distinguishing portfolio demo and then manages to talk intelligently about C++ at interview.

THE APPRENTICESHIP

Unfortunately, selecting the right candidate for a graduate position is often more difficult and time consuming than employing someone already in the industry. When a programmer has worked for another studio and produced published work, the risks are much slighter. Particularly as the games industry is small enough to make it easy to contact a former colleague who can vouch for their abilities.

Studios always prefer to recruit staff with at least one published title to their name, and in some cases would rather leave a post unfilled than recruit a new graduate. This may seem self-defeating, but recruitment often only takes place when a project is already behind schedule and putting a graduate with no track record onto an over-stretched project simply compounds the problem. Obtaining a published title is informally seen as the mark of having completed a form of apprenticeship in game development, suggesting (in principle) that a programmer is able to make a useful contribution to a project. Nonetheless it can potentially take up to 18 months to develop a commercial game, so it is not easy to earn this status.

Graduates lucky enough to get offered a position will often arrive onto a busy project with the expectation that they can teach themselves new skills as they go along. Ironically, they are unlikely to be let anywhere near the parts of the game most closely linked to the technology-based demos they produce in their student portfolios. These aspects are all handled by the senior programmers. They are far more likely to be given self-contained tasks that have a limited potential impact on the rest of the game’s code-base such as tools, plug-ins, menu-systems, mini-games, particle effects or sound.

Part of the reason that graduates are kept away from the architecture of the game is that they rarely arrive with a practical understanding of how to expand and work within a large code structure. To some extent different studios
(and even different teams within a studio) have different ideas about what an ideal game architecture looks like, but it is very easy for an inexperienced programmer to create additional work for their senior colleagues by adopting a naive approach. Thus they are kept upon the periphery of a project and probably won’t be allowed to participate in higher-risk activities until they have completed their ‘apprenticeship’.

So herein lies the paradox. Students can’t get a job in the games industry without a published title to their names, but can’t get a published title without working in the games industry.

EDUCATING A GRADUATE

So how does this problem look from the other side of the fence? How do universities running game development courses equip students for this kind of role, and what steps can academia take to try and get students out of this Catch 22 situation?

Sheffield Hallam University runs a portfolio of undergraduate and postgraduate degrees based around game development. My own focus is on the Games Software Development BSc and MComp degrees, which are designed primarily for programmers aspiring to enter the games industry. These courses have a strong software-engineering element revolving around the very C++ skills that the console games industry requires. In the 3-year BSc programme these students are subjected to an intensive programme of modules in the fields of computer architecture, programming languages, object-oriented design, mathematics, 3d graphics, asset management and game prototyping. The four-year MComp then expands on this with modules on tools, libraries and frameworks, games case studies and a group project.

By the end of the third year students have been exposed to the complete range of game programming skills, and have all the separate areas of specialist knowledge they need to create a complete game. However many of them still struggle to do so, resulting in only a very small number of completed C++ games in graduate portfolios.

THE DEMANDS OF 3D GAME PROGRAMMING

Creating a 3D game in C++ from scratch requires an overwhelming range of understanding and skills. These range from practical programming skills using compilers and debuggers with specialist hardware and API’s, to a more theoretical understanding of 3D mathematics, computer architectures and object-oriented design. And all this before you even start to think about gameplay, resource pipelines, development methodologies, team-working tools or working within fixed memory, storage and processing constraints. Moreover, a developer doesn’t just need to understand these in isolation (as they might be taught in a university course), but has to bring them all together into a finished product as part of a multi-disciplinary team.

It’s easy for experienced game programmers to take these skills for granted, but many of us gained them over a period of decades—not just three years. Games themselves were much less complicated when we began making them, and they were developed by much smaller teams. So you could argue that in many ways it is completely unrealistic to expect a graduate programmer to take on the task of writing a complete 3D game in C++.

ENGINE SOLUTIONS

In light of the incredible demands of the task, one solution which Universities have explored is to use existing game engines. Professional studios rarely write the entire codebase for each game from scratch and will usually start with an existing C++ game engine which already provides a range of key functionality such as rendering, collision and physics. Such engines
might be created in-house or licensed as off-theshelf technology from third party companies. The latter are also potentially available to Universities as an authentic solution to the problem. Of course the professional ones are extremely expensive, and the cheaper alternatives tend to lack the documentation and supporting toolsets. They are also very complicated in their own right, as a result of being designed around their features and performance rather than ease of understanding. This can actually make it as hard for a student to get to grips with an engine as it would be to create a simple one of their own.

More recently, game engines such XNA, Unity and the Unreal UDK have gained more traction educationally. All provide students with the potential to produce commercial-quality results within limited timeframes. All are free and have thriving hobbyist support communities around them. However, none are based on C++, and the industry would actually see the latter two as sophisticated game editors, rather than engines which makes them the domain of designers—not programmers. XNA has more credibility programming-wise, but uses C# rather than C++. C# is very similar to C++ (and is used increasingly for tool development in the games industry), but it is a step in the wrong direction from the perspective of a game studio that already sees graduates as lacking in industry readiness.

THE STATE OF PLAY

So it is unsurprising that in my previous job I failed to see the quantity of fully-fledged C++ gaming projects that I would have liked in student portfolios. University courses provide students with a lot of skills, but these do not result in the level of project readiness that the industry seeks. The period of apprenticeship required by graduates will therefore continue and so will the disincentive for the industry to employ them.

This apprenticeship has obvious parallels with Lave and Wenger’s work on situated learning and the role of legitimate peripheral participation in becoming part of a professional community of practice. The games industry is by no means unique in attempting to manage risk by getting new graduates to participate peripherally on projects. So there follows a few ways in which I have attempted to try and extend peripheral participation beyond the boundaries of a game studio and bring situated learning into the classroom.

WORK WITH EXISTING CODEBASES

A codebase is the focus of the distributed knowledge and shared practices of a game programming team. Ultimately it is the participation of junior team members on peripheral tasks within a codebase that forms the basis of their apprenticeship. If graduates cannot get into studios to participate on a codebase then perhaps codebases can get out of studios instead. In fact the code to classic games like Wolfenstein, Doom and Quake have already been made public by their developers. They are free to download, and represent an evolution of practice that has led to more contemporary games like Unreal Tournament—including the critical shift from 2D to 3D. Modifying these codebases to support new features, producing additional tools which improve aspects of the pipeline, or simply porting the code to new platforms all represent tasks which would allow students to participate legitimately around these codebases.

My former employers were gracious enough to let me take one of my project codebases into the classroom, in the form of the cross-platform side of a prototype game for the Wii (Outnumbered). My MComp students were able to use their existing knowledge of DirectX programming to port the codebase to run on the PC as their group project for this year. This included replacing the rendering, collision and sound systems, as well as adding a new front-end to the game. This was all framed around an agile development model,
where the students worked in regular sprints to deliver functionality agreed by them at the start of the sprint. All of this provided a very authentic contact with a real codebase, but yet was achievable within the teaching and assessment constraints of the module.

LESS EMPHASIS ON TECHNOLOGY

The fact is that while games clearly rely on cutting-edge technology, new graduates are more likely to be asked to work with and around this technology than engineer it. A graduate who can write an impressive shader, but doesn’t know how to load data from a binary file is clearly learning to run before they can walk. Nearly all of the lessons learned from 2D game development in C++ are relevant to 3D and yet the demands of the task are much more achievable. The complicated architecture of modern, multi-platform games has evolved through the experience of developing simpler games and students can only truly appreciate the end-point by seeing the evolution.

The game ported to the PC by the MComp students was fundamentally 2D in operation, and contained little in the way of cutting-edge technology. This meant it didn’t require a sophisticated game engine, and it was far easier for students to see how each aspect of the code worked together to produce the game. The simplicity of the game meant that the task of porting it was achievable and with a limited amount of support I was able to keep it within their “zone of proximal development”.

AUTHENTIC PROJECT DOCUMENTS

Codebases are not the only visible outputs of a studio’s community of practice. Most projects also produce documentation in the form of game designs, technical designs and style guides as well as project schedules/plans and meeting minutes. All of these provide an additional insight into the community of a studio. Before I left the industry to take my PhD I worked for Infogrames in the UK and was given permission to use documentation and examples from my former projects in my teaching. Hogs of War and Micromachines (PS2/Xbox/GC) were projects with very different approaches and very different outcomes. My MComp students were able to read the development documentation for themselves and even play versions of Hogs at different points in its development. They were then able to look at how the games were received in the press and contrast the approaches taken. This gave them a completely different kind of insight into the community of a game development studio, but equally as valuable as the exposure to a codebase.

GAME STUDIO/COMPETITIONS

One of the most direct ways to provide situated learning in game development is simply to put a game studio in your university. In fact some are already doing this and we even have plans for one of our own in the future. Indeed, Sony are actively encouraging universities to become developers and publishers of console games on platforms like the PSP. Six months developing and releasing a hand-held title for the PSP is the kind of experience that would seriously bolster a student’s CV.

Nonetheless you don’t necessarily need your own studio for students to be exposed to this kind of experience. Competitions such as “Dare to be Digital” here in the UK offer student teams the chance to take part in a 3-month development experience working on a game prototype. This year Sheffield Hallam had two teams succeed in making the last 16 of this international game development competition. Our MComp students will be applying their game development skills in an attempt to prototype a game on the PSP in just 10 weeks.
CONCLUSION

Students take game development degrees in the hope that it will get them a foot in the door of the games industry. Nonetheless, the industry doesn’t see them as the finished product and often won’t employ them without experience of one published title.

Universities could improve the employment prospect of their students within the console-game sectors by providing more opportunities for situated learning during their degree courses. Working with existing code-bases, a switch of emphasis from technology to architecture, and more student studios and competitions are potential ways of providing this.

Finally developers also need to wake up to their own responsibility in solving this problem. If they are unprepared to take on inexperienced staff then they need to be more open in working with universities to help to provide authentic learning experiences within an academic environment. A greater willingness to release codebases and authentic project resources would help to ensure a more industry-ready workforce in the future.

M. P. Jacob Habgood
Guest Editor
JGCMS

M. P. Jacob Habgood is a Senior Lecturer in Game Development at Sheffield Hallam University in the UK. Prior to this his games industry career spanned 14 years and includes a dozen published titles for all the major console platforms. He studied his Ph.D. in game-based learning at The Learning Sciences Research Institute at the University of Nottingham and was awarded his doctorate in 2007. Jacob is also the author of a popular series of books on hobbyist game development, beginning with “The Game Maker’s Apprentice”.