Desafios no Ensino de Programação a Alunos de Videojogos.

Nuno Fachada

Universidade Lusófona de Humanidades e Tecnologias, Lisboa

nuno.fachada@ulusofona.pt

Resumo – As unidades curriculares (UCs) de programação num curso de Videojogos devem ter dois objetivos: 1) um objetivo mais geral, que consiste em fornecer aos alunos as bases que lhes permitam assimilar conceitos gerais de programação, matemática e física, bem como desenvolver o seu pensamento lógico e algorítmico; e, 2) um objetivo mais específico, que consiste na aprendizagem das ferramentas e conceitos concretos que permitam aos alunos trabalhar de forma fluída no *game engine* de eleição do curso. Este último objetivo enquadra as UCs de programação numa lógica *top-down*, pois a seleção do *game engine* guia a forma como os respetivos programas são preparados. Dentro desta perspetiva, as UCs de Programação devem alimentar e ter em vista possíveis colaborações com as restantes UCs, em especial as de *game development* puro. De forma transversal, os exemplos de aula e projetos de avaliação devem estar devidamente adaptados aos alunos em questão, de modo a tornar a exposição das matérias o mais apelativa possível. Neste documento discutiremos a forma como estes desafios estão a ser abordados na Licenciatura em Videojogos da Universidade Lusófona de Humanidades e Tecnologias.

1. Introdução

Os Videojogos são produtos complexos que incorporam *software*, *design* e arte. Como tal, seu desenvolvimento é um esforço transdisciplinar, que tipicamente requer profissionais de diversas áreas, de preferência com conhecimentos nos diferentes campos de ação (Bates, 2004; Sturtevant et al., 2008; Yee, Sturman, & Feiner, 2007). Devido à variedade de conhecimentos necessários, e com o propósito de formar profissionais qualificados, têm surgido desde o final dos anos 90, início do século, vários cursos superiores de *design* e desenvolvimento de Videojogos, em particular na América do Norte e na Europa (Ficocelli & Gregg, 2005; Jones, 2000; Parberry, Roden, & Kazemzadeh, 2005; Zyda, 2006; Zyda, Lacour, & Swain, 2008). Recentemente, e seguindo esta tendência, vários destes cursos surgiram também em Portugal (Sociedade Portuguesa de Ciências dos Videojogos, 2019).

A programação é uma área fundamental no ensino em Videojogos. A programação foi, inclusive, a única competência necessária para a criação dos primeiros jogos (Reimer, 2005; Sturtevant et al., 2008). No entanto, o aparecimento dos *game engines* e outras ferramentas de desenvolvimento tem vindo a reduzir a importância relativa da programação – de tal forma que hoje em dia é possível desenvolver jogos comercialmente viáveis, embora limitados, sem escrever código (Astle-Adams, 2018; Cawley, 2015; Dalal, Dalal, Kak, Antonenko, & , 2009; Peng, 2015). Em todo o caso, a

programação continua a ser uma competência essencial para a criação de jogos interessantes e originais. De forma mais geral, as Ciências da Computação, permanecem como uma componente central num curso de desenvolvimento de Videojogos.

A programação de jogos, como os próprios jogos, é complexa e possui várias sub-especializações. Alguns exemplos são a computação gráfica, simulação de física, programação multi-tarefa, programação de baixo nível para otimização, programação orientada a objetos, inteligência artificial, bases de dados e assim por diante (Burns, 2008; Claypool & Claypool, 2005; Fachada, 2018; Herbert, de Salas, Lewis, Dermoudy, & Ellis, 2014; Parberry, 2011). Desta forma coloca-se a questão: como pode a programação de Videojogos, com tantas especializações e particularidades, ser adequadamente lecionada numa Licenciatura em Videojogos? Em primeiro lugar, é necessário definir de forma mais concreta o tipo de Licenciatura que se pretende oferecer. Segundo Mateas e Whitehead (2007), os cursos de graduação em Videojogos podem ser organizados de três formas:

1. Orientados para as Ciências da Computação, *B.Sc.*
2. Orientados para a Arte e/ou *Design*, *BA*
3. Uniformemente interdisciplinares / banda larga, *B.Sc.* ou *BA*

Relativamente aos cursos uniformemente interdisciplinares, os mesmos contêm, de acordo com os mesmos autores, uma "forte base de Ciências da Computação", apesar de "não aprofundarem tanto esta área como o primeiro tipo de cursos", oferecendo em substituição "um conjunto mais amplos de disciplinas de *game design*". Ainda de acordo com estes autores, os programas interdisciplinares são os "mais desafiantes de implementar".

A Licenciatura em Videojogos da Universidade Lusófona de Humanidades e Tecnologias (ULHT) é um BA (*Bachelor of Arts*) uniformemente interdisciplinar, de banda larga, com a duração de 3 anos. Trata-se de um curso acessível a partir diversas áreas científicas do ensino secundário. Em consequência, alunos de áreas como as Artes e das Humanidades poderão ter eventuais e naturais dificuldades nas unidades curriculares (UCs) das Ciências da Computação. Além disso, o curso é apetecível para alunos com grande paixão pelos Videojogos na sua vertente lúdica – algo que não se reflete necessariamente no gosto pelo desenvolvimento e produção dos mesmos.

Definido o tipo de Licenciatura que a ULHT oferece, podemos agora refinar a questão a anteriormente colocada da seguinte forma: como podem as Ciências da Computação em geral, e a programação em particular, serem lecionadas com sucesso num BA uniformemente interdisciplinar de três anos? Neste documento discutiremos a forma como estes desafios estão a ser abordados na Licenciatura em Videojogos da ULHT. Em concreto, na Secção 2 são apresentadas três estratégias utilizadas para este propósito (bem como as referências que as suportam), nomeadamente: 1) desenvolvimento curricular holístico em sentido *top down*; 2) colaboração entre UCs de programação e UCs de desenvolvimento de jogos; e, 3) utilização de jogos como projetos nas UCs em análise. Os

resultados preliminares destas abordagens são discutidos na Secção 3. As conclusões são debatidas na Secção 4.

2. Estratégias

Neste trabalho, e com o propósito de responder à questão colocada na Introdução, propomos que as unidades curriculares (UCs) de programação num curso desta índole tenham dois objetivos:

1. Um objetivo mais geral, que consiste em fornecer aos alunos as bases que lhes permitam assimilar conceitos gerais de programação, matemática e física, bem como desenvolver o seu pensamento lógico e algorítmico (Futschek, 2006).

2. Um objetivo mais específico, que consiste na aprendizagem das ferramentas e conceitos concretos que permitam aos alunos trabalhar de forma fluída no *game engine* de eleição do curso (Dickson, 2015).

O segundo objetivo enquadra as UCs de programação numa lógica *top-down*, pois a seleção do *game engine* guia a forma como os respetivos programas são preparados. Na Subsecção 2.1 descrevemos a forma integrada como o programa curricular das UCs de programação foi implementado. Dentro desta perspetiva, as UCs de programação devem alimentar e ter em vista possíveis colaborações com as restantes UCs, em especial as de *game development* puro. Alguns casos concretos deste tipo colaborações na Licenciatura em Videojogos da ULHT são discutidos na Subsecção 2.2. De forma transversal, os exemplos de aula e projetos de avaliação devem estar devidamente adaptados aos alunos em questão, de modo a tornar a exposição das matérias o mais apelativa possível. As abordagens adotadas neste particular são apresentadas na Subsecção 2.3.

2.1 Desenvolvimento curricular holístico *top-down*

Tal como realçado na Introdução, as Ciências da Computação estão omnipresentes no desenvolvimento de jogos de computador. De modo a implementarmos um programa coerente na área dos Videojogos é em primeiro lugar necessário tomar algumas decisões chave, que por sua vez moldarão o restante desenho curricular. Em concreto, é necessário definir (Parberry et al., 2005):

1. Quais os tópicos avançados que os alunos devem dominar?

2. Que *game engine* utilizar?

As duas decisões não são totalmente independentes, uma vez que o *game engine* pode incluir suporte ou facilitar a aprendizagem dos tópicos avançados. Desta forma, as duas decisões devem ser tomadas em conjunto. Em todo o caso, o restante currículo deve ser construído em torno destas duas escolhas, de forma *top-down*. Nos pontos 2.1.1 e 2.1.2 apresentaremos, de forma justificada, estas duas decisões chave. Nos pontos seguintes, de 2.1.3 a 2.1.5, discutiremos a elaboração do restante

currículo com base nestas decisões, nomeadamente no que concerne à escolha das linguagens de programação a lecionar no curso, bem como à seleção das matérias de suporte. Finalmente, no ponto 2.1.6, apresentaremos a implementação prática do programa com base nestas escolhas.

2.2.1. Seleção de tópicos avançados

Existem vários tópicos avançados das Ciências da Computação diretamente aplicáveis ao desenvolvimento de Videojogos, mas que dificilmente poderão ser lecionados nas UCs nucleares da área, sobretudo num BA de 3 anos uniformemente interdisciplinar, como é o caso da Licenciatura em Videojogos da ULHT. Alguns destes tópicos são:

- Computação Gráfica (Coleman, Krembs, Labouseur, & Weir, 2005)
- Teste e Validação de *Software* (Claypool & Claypool, 2005; Leutenegger & Edgington, 2007)
- Computação em Rede e/ou Distribuída (Ip, 2012; Parberry et al., 2005; Sweedyk, deLaet, Slattery, & Kuffner, 2005)
- Computação Paralela (Sweedyk et al., 2005)
- Bases de Dados (Fachada, 2018)
- Inteligência Artificial (Millington, 2019; Roden & LeGrand, 2013)

Todos estes tópicos são importantes e merecedores de uma UC própria. No entanto a lecionação completa dos mesmos apenas é possível num curso de 4 ou 5 anos orientado para as Ciências da Computação. Num curso como o da ULHT, haverá espaço para um, no máximo dois tópicos – isto se forem consideradas UCs opcionais.

O curso teve até ao ano letivo 2018/19 uma UC de Bases de Dados. Contudo, esta UC nunca foi muito popular entre os alunos, e mesmo após especial esforço por parte dos autores (Fachada, 2018), a sua aplicabilidade nos jogos desenvolvidos no âmbito do curso foi sempre limitada.

No ano letivo 2017/18, o tópico de Inteligência Artificial foi introduzido no 3.º ano do curso sob a forma de uma UC opcional com o mesmo nome. Devido à sua aplicabilidade direta em praticamente todo o tipo de jogos, a sua aceitação e interesse por parte dos alunos tem sido superior, por exemplo, à verificada na UC de Bases de Dados. Numa série de alterações curriculares recentes, optámos por suprimir as UCs de Base de Dados e Inteligência Artificial (ambas com 4 ECTS), tendo sido criada uma nova UC obrigatória denominada Sistemas Artificiais e Emergentes, com 6 ECTS e maior abrangência na temática de Inteligência Artificial para Videojogos. O tópico de Bases de Dados não desaparece completamente, pois é lecionado de forma mínima nas UCs nucleares, como discutido no ponto 2.1.3.

Em conclusão, e tendo em conta os argumentos apresentados nesta discussão, a Inteligência Artificial para Videojogos é neste momento a única temática avançada de Ciências da Computação no curso de Videojogos da ULHT com uma UC própria.

2.2.2. Escolha do *game engine*

Um *game engine* deve facilitar e acelerar o processo de desenvolvimento, permitindo aos alunos a criação de jogos interessantes num curto espaço de tempo (e.g., um semestre ou um ano). Tipicamente é esperado que o *game engine* suporte renderização 2D e 3D de alta qualidade, efeitos especiais, simulação de física, animações, reprodução sonora, comunicações de rede, entre outras capacidades importantes no desenvolvimento de Videojogos (Wu & Wang, 2012). Autores como Wu e Wang (2012) e Dickson, Block, Echevarria, e Keenan (2017) fornecem algumas diretrizes para a escolha de um *game engine* para ensino. Entre estas destacam-se o custo (ou ausência do mesmo), qualidade da documentação, flexibilidade e extensibilidade, suporte para múltiplas plataformas, curva de aprendizagem e estabilidade de execução.

A tarefa de encontrar o *game engine* mais adequado para determinado curso não é trivial, e são vários os relatos de dificuldades nesta escolha (Dickson, 2015; Dickson et al., 2017; Parberry et al., 2005; Peng, 2015). Vários são também os relatos sobre o Unity *game engine* (Unity Technologies, 2018) como uma escolha razoável dentro das diretrizes estabelecidas no parágrafo anterior: custo zero, boa documentação, flexível e extensível, com uma variedade de extensões disponíveis na respetiva *asset store*, multi-plataforma (tanto a nível do editor, como a nível das plataformas para as quais os jogos podem ser exportados), *scripting* com uma linguagem de programação muito versátil (C#), curva de aprendizagem não muito acentuada, e, acima de tudo, possibilidade de criação de jogos "a sério" (Dickson, 2015), sendo amplamente usado na indústria. O Unity é inclusivamente usado ao nível do ensino secundário (Comber, Motschnig, Mayer, & Haselberger, 2019). Devido a estes fatores, o Unity é o *game engine* de eleição na Licenciatura de Videojogos da ULHT.

Um possível problema com o Unity é o seu rápido desenvolvimento, que torna a literatura disponível rapidamente obsoleta (Dickson, 2015), incluindo os próprios videos e tutoriais providenciados pela Unity Technologies. Contudo, o manual de utilizador e API estão geralmente atualizados, minimizando o problema, mas obrigando os docentes a estarem constantemente a par das novidades (Ip, 2012) – o que de certa forma até ajuda a manter os cursos relevantes.

2.1.3. A linguagem C# e respetivas implicações ao nível das matérias lecionadas

A linguagem de programação C# A principal linguagem de programação utilizada no curso de Videojogos da ULHT é o C#, uma vez que se trata da linguagem utilizada pelo Unity para *scripting*. Trata-se de uma linguagem de programação orientada a objetos, de nível intermédio, com gestão automática da memória. A linguagem suporta ainda os paradigmas de programação funcional e orientada a eventos, bem como gestão manual de memória em casos específicos. Resumindo, é uma linguagem bastante versátil (Albahari & Albahari, 2017).

Adicionalmente, o C# é utilizado como linguagem de *scripting* numa série de *game engines* além do Unity, e.g., Godot (Linietsky & Manzur, 2019), CryEngine (Crytek, 2019), Xenko (Silicon Studio, 2019), MonoGame (Williams & Spilman, 2019) ou Unigine (Unigine Corp, 2019). A sintaxe é

praticamente igual à da linguagem Java, e semelhante à do C e C++. Desta forma, a estudo do C# não limita os alunos, antes pelo contrário: oferece uma base sólida para futura aquisição de conhecimento no âmbito da programação de Videojogos.

Uma possível limitação desta escolha reside no facto da linguagem *standard* da industria de Videojogos ser o C++ (Hewner & Guzdial, 2010; Ip, 2012). O C++ é também a linguagem usada para desenvolvimento em Unreal Engine (Epic Games, 2019), em si um *standard* da industria no que concerne a jogos AAA. Contudo, existem boas razões para evitar a linguagem C++ num curso de 3 anos uniformemente interdisciplinar, como é o caso em estudo. Trata-se de uma linguagem enorme e complexa, com múltiplas versões e com diferentes formas de realizar tarefas comuns (Millington, 2019), sendo difícil de dominar num curso com esta duração. Existe inclusive um relato bastante negativo do uso de C++ e Unreal Engine para ensino, no qual os autores concluem que o uso de *blueprints*[8]acaba por ser a única forma viável para criação de jogos num curto espaço de tempo. O problema torna-se ainda menos relevante se considerarmos que: a) a componente de herança da orientação a objetos do C# é fortemente baseada na sua correspondente do C++; e, b) o C++ tem como base a linguagem C, bastante mais simples e também lecionada no curso, como referido no ponto 2.1.4. Consequentemente, acreditamos que o programa aqui discutido prepara os alunos da melhor forma para uma possível posterior transição para o C++.

Implicações ao nível das matérias lecionadas A escolha do C# tem implicações importantes ao nível das matérias lecionadas nas UCs nucleares de Ciências da Computação. A orientação a objetos é em si um paradigma de programação sobre o qual os alunos têm de aprender a raciocinar. É geralmente difícil de determinar o nível de abstração a utilizar, em particular a quantidade de classes e as relações entre elas. Em conjunto com a gestão automática de memória, a orientação a objetos afasta o programador do *hardware*. É perfeitamente possível alguém ter um bom conhecimento da linguagem C#, nomeadamente da sua sintaxe e principais classes, e ser simultaneamente incapaz de produzir projetos arquiteturalmente sólidos e com bom desempenho, duas características fundamentais no desenvolvimento de Videojogos (Nystrom, 2014).

Desta forma, torna-se crucial que o ensino da programação por objetos a um nível mais amplo e detalhado seja tido em conta nos programas a desenvolver. Em particular, é nossa opinião que sejam lecionados princípios fundamentais da programação por objetos (e.g., SOLID e GRASP), bem como *design patterns*. Relativamente a estes últimos, o foco deve estar nos *design patterns* para jogos, que têm em consideração otimizações importantes para Videojogos (Claypool & Claypool, 2005; Hewner & Guzdial, 2010; Nystrom, 2014; Sweedyk & Keller, 2005). Naturalmente, é também importante a inclusão de diagramas UML de classes no currículo, pois estes são a melhor forma de comunicar visualmente *designs* de classes (Claypool & Claypool, 2005; Leutenegger & Edgington, 2007; Sweedyk & Keller, 2005).

[8] Sistema visual de *scripting* do Unreal Engine, que dispensa o uso de programação C++.

A escolha do C# como principal linguagem a lecionar no curso apresenta ainda uma vantagem importante com implicações mais amplas ao nível da seleção dos tópicos avançados discutidos no ponto 2.1.1. Em concreto, a linguagem C# oferece as ferramentas LINQ (*Language Integrated Query*) que permitem realizar pesquisas em estruturas de dados e bases de dados ao estilo SQL de forma nativa no código. Consequentemente, a alocação de algumas horas de aulas para aprendizagem das ferramentas LINQ minimiza parcialmente potenciais impactos negativos causados pela supressão da UC de Bases de Dados (referida no ponto 2.1.1).

2.1.4. Linguagens para aprender programação

A linguagem C#, apesar de mais simples do que o C++, envolve uma série de paradigmas e características que a tornam pouco apelativa para a aprendizagem da programação a partir do zero. Desta forma, e até para expandir os horizontes dos alunos, devem ser lecionadas outro tipo de linguagens mais simples no início do curso. Propomos que os alunos aprendam duas linguagens diametralmente opostas em duas UCs distintas: uma linguagem de baixo nível, próxima do *hardware*, e outra de alto nível, para *scripting* e com elevado nível de abstração. A primeira deverá ser ensinada no contexto e em paralelo com conceitos básicos de arquitetura de computadores, onde poderá inclusive ser necessário manipular informação a nível binário. A segunda deverá ser lecionada tendo em vista a rápida criação de aplicações (e.g., jogos, ver Subsecção 2.3), promovendo o desenvolvimento do pensamento lógico e algorítmico dos alunos (Futschek, 2006). O objetivo é que os alunos entendam o que está em jogo quando começarem a utilizar uma linguagem de nível intermédio tal como o C#. Por outras palavras, pretende-se que os alunos compreendam o custo das abstrações utilizadas e consigam identificar e eliminar problemas de desempenho.

Escolhemos a linguagem C como linguagem de programação de baixo nível para ensino de programação no 1.º semestre do curso. Como contraponto, optámos pela linguagem Python para ensino de programação e prototipagem de alto nível. A justificação destas escolhas está detalhada nos dois parágrafos seguintes.

C como linguagem de baixo nível

A linguagem C é simples, expressiva e eficaz, sendo suportada em virtualmente todo o tipo de plataformas (Katz, 2013; Mortoray, 2012). É a língua franca da programação de sistemas (Katz, 2013; Spolsky, 2005), e a sua sintaxe é base de muitas outras linguagens, como por exemplo C++, C# e Java. No caso do C++ a relação ainda é mais forte, uma vez que o C é essencialmente um subconjunto do C++. O facto de ser uma linguagem muito próxima do *hardware* ajuda na compreensão de como um computador funciona, permitindo aprender a realizar todo o tipo de otimizações de desempenho, qualidade muito procurada pela industria de jogos (Hewner & Guzdial, 2010). Finalmente, e apesar do C ser uma linguagem de excelência para o ensino da programação no geral, a existência de bibliotecas

orientadas para os Videojogos, casos de Allegro[9], RayLib[10] ou SDL[11], abre ainda mais as portas do C para o caso particular da programação de Videojogos.

Python como linguagem de alto nível

Considerada como uma excelente linguagem para introdução à programação (Cleary, Vandenbergh, & Peterson, 2015), o Python apresenta uma sintaxe simples, cujos blocos são definidos pela indentação do código, de forma totalmente oposta ao que acontece no C ou C#. Estamos em crer que esta característica obriga os alunos a entender a importância da indentação, além de lhes abrir os horizontes relativamente a diferentes tipos de sintaxe. A quantidade e qualidade das bibliotecas disponíveis para Python é extraordinária, permitindo a rápida prototipagem de todo o tipo de *software*, incluindo naturalmente, Videojogos. Adicionalmente, o Python é utilizado como linguagem de *scripting* numa série de *engines* (e.g., Blender, Cocos2d, Irrlicht, Maya, Panda3D), bem como de jogos propriamente ditos (e.g., Battle for Wesnoth, Civilization IV, CodeSubWars, Minecraft).

2.1.5. Matemática e Física para jogos

A Matemática e a Física são componentes fundamentais no desenvolvimento de Videojogos: praticamente todos os jogos incorporam modelos matemáticos e/ou simulações de física (Coleman et al., 2005). Exposições detalhadas destas matérias estão disponíveis numa variedade de livros e publicações, entre quais destacamos a abordagem acessível e informal proposta por Dunn e Parberry (2011), bem como a metodologia abrangente e detalhada apresentada por Lengyel (2012).

No programa em discussão temos duas UCs dedicadas exclusivamente ao ensino de Matemática e Física para jogos (referidas no próximo ponto), cujo programa é baseado no livro de Dunn e Parberry (2011). Na primeira UC, os exercícios, exemplos e projetos são implementados em Python (linguagem que está a ser lecionada simultaneamente numa UC de programação), usando as bibliotecas PyGame (McGugan, 2007), NumPy (van der Walt, Colbert, & Varoquaux, 2011) e Matplotlib (Hunter, 2007). Na segunda UC, o Python continua a ser usado, sendo a dada altura introduzidos o C# e o Unity, de modo a que os alunos possam contextualizar a matéria lecionada no âmbito do *game engine* utilizado no curso.

2.1.6. Implementação prática

A Figura 1 mostra, ao nível das UCs, os conteúdos programáticos, discutidos nos pontos anteriores, contextualizados nos quatro primeiros semestres do curso.

O 1.º semestre tem uma forte componente de Ciências da Computação, com três UCs dedicadas. Os alunos aprendem a programar usando C e Python, dentro da lógica discutida no ponto 2.1.4, sendo

[9] https://liballeg.org/
[10] https://www.raylib.com/
[11] https://www.libsdl.org/

simultaneamente lecionada a matéria introdutória de Matemática e Física para jogos, usando a linguagem Python para prototipagem. A Matemática e Física para jogos continuam a ser desenvolvidas no 2.º semestre, embora com conceitos mais avançados. Em paralelo é introduzida a programação por objetos em C#, linguagem que começa também a ser utilizada em Unity numa UC de desenvolvimento de jogos. O projeto de 1.º ano é produzido, em grande parte, nesta UC.

No 3.º semestre (2.º ano), a carga horária de Ciências da Computação desce consideravelmente, com apenas uma UC exclusivamente dedicada ao tópico. Nesta UC são discutidos conceitos avançados de programação por objetos (e.g., *design patterns*). O objetivo é que no final deste semestre os alunos sejam proficientes não só em C#, como também em *design* estruturado e computacionalmente eficiente de classes.

Finalmente, no 4º semestre, é oferecida uma UC na qual são discutidos diferentes tópicos de Inteligência Artificial para jogos. Esta UC é a última com conteúdos exclusivos das Ciências da Computação.

Como é possível observar na Figura 1, existem diversas oportunidades de colaboração entre UCs, algumas das quais são discutidas na subsecção seguinte.

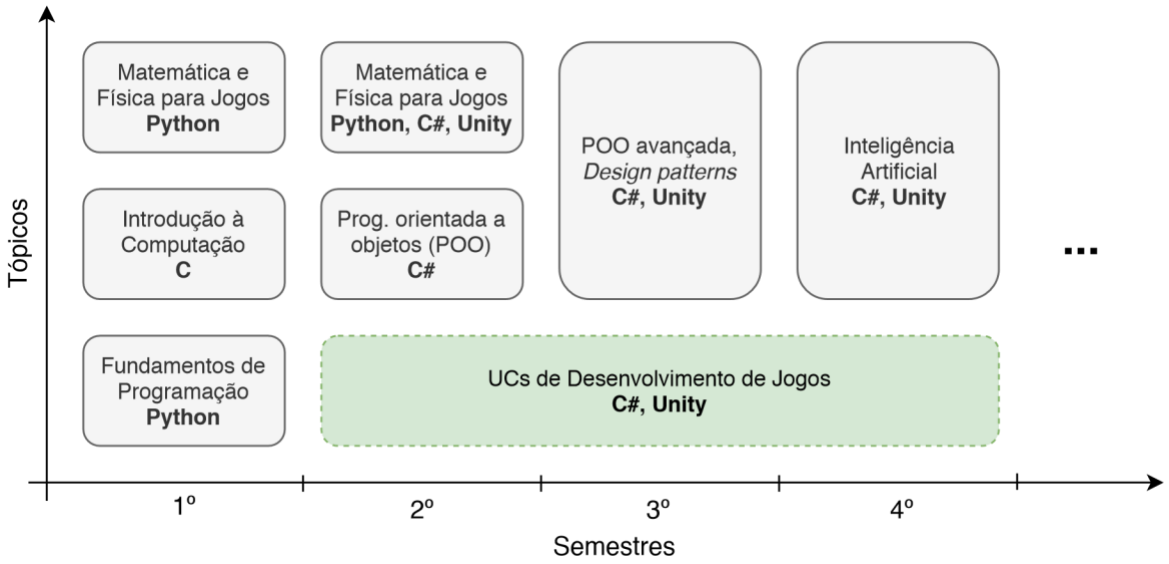

Figura 1 – Implementação prática do currículo das UCs de Ciências da Computação da Licenciatura em Videojogos da ULHT. Caixas sólidas (a cinzento) correspondem a UCs de Ciências da Computação, com indicação dos conteúdos programáticos. A caixa a tracejado (verde claro) engloba várias UCs de desenvolvimentos de jogos ao longo dos semestres.

2.2. Colaboração com UCs de desenvolvimento de jogos

Em geral, o projeto semestral ou anual da(s) UC(s) de desenvolvimento de jogos consiste na criação de um Videojogo em todas as suas vertentes: *game design*, programação, desenho 2D/modelação 3D,

animação, som e por ai em diante. As UCs de Ciências da Computação podem colaborar com as de desenvolvimento de jogos de duas formas:

1. A componente de computação do projeto (programação, *design* de classes, bases de dados, inteligência artificial, etc) é avaliada na respetiva UC de Ciências da Computação.
2. A componente de computação do projeto (programação, *design* de classes, bases de dados, inteligência artificial, etc) é desenvolvida na respetiva UC de Ciências da Computação.

Até ao momento implementámos os seguintes casos concretos na Licenciatura em Videojogos da ULHT, ambos no 1.º semestre do 2.º ano (2018/19):

1. A componente de programação do projeto da UC *Desenvolvimento de Jogos Digitais II* foi considerada como elemento de avaliação da UC *Linguagens de Programação II*.
2. A componente de bases de dados do projeto da UC *Desenvolvimento de Jogos Digitais II* foi desenvolvida na UC *Bases de Dados*.

No primeiro caso foi avaliada a qualidade e organização do código, o *design* de classes, indentação, quantidade e qualidade de comentários, bem como a documentação ao nível do código. No segundo caso, o projeto consistiu na implementação de um jogo 3D na primeira pessoa com inventário de itens colecionáveis. O inventário foi inicialmente implementado na UC *Desenvolvimento de Jogos Digitais II* com recurso a estruturas de dados nativas em C#, sendo posteriormente substituído por uma base de dados em SQL Server, desenvolvida na UC de *Bases de Dados*.

2.3. Videojogos como projetos e exemplos

A utilização de Videojogos como projetos e exemplos de aula tem sido ao longo dos anos uma estratégia clássica para aumentar o interesse e o número de matrículas em cursos de Ciências da Computação (Coleman et al., 2005; Jones, 2000; Leutenegger & Edgington, 2007; Morrison & Preston, 2009; Peng, 2015). Os jogos fornecem ainda um excelente contexto para o ensino de *design patterns*, balizando um tópico de natureza abstrata e por vezes difícil de lecionar (Astrachan, 2001; Connolly, 2007; Wick, 2005).

Em cursos de desenvolvimento de Videojogos esta abordagem é praticamente obrigatória, de modo a não defraudar as expetativas dos alunos. Desta forma, a exposição dos diferentes tópicos das Ciências da Computação torna-se mais apelativa.

Esta abordagem tem sido utilizada nos últimos dois anos letivos na ULHT, tanto a nível de exercícios e exemplos de aula, bem como ao nível dos projetos de avaliação. Por exemplo, em 2018 propusemos uma série de exercícios de Bases de Dados adaptados para alunos de Videojogos (Fachada, 2018). Em

2017 criámos a página de GitHub *VideojogosLusófona*[12], na qual estão presentes e vão sendo colocados enunciados de projeto, exemplos e exercícios de aula orientados para os alunos do curso. A nível de projetos, temos propostas como a reimplementação de videojogos clássicos, a implementação computacional de diferentes jogos de tabuleiro[13], e até alguns enunciados originais – sem dúvida os mais complexos de elaborar por parte dos docentes.

3. Resultados preliminares

As estratégias descritas na secção anterior começaram a ser colocadas em prática no ano letivo 2017/18, sendo possível apresentar alguns resultados preliminares. Num curso de Videojogos os resultados podem ser analisados com base na qualidade dos jogos produzidos nas UCs de desenvolvimento de jogos, bem como na evolução técnica dos alunos em comparação com anos anteriores. Nesta perspetiva, realçamos cinco resultados fundamentais:

- Aumento do número de projetos de alta qualidade
- Aumento da qualidade média dos projetos
- Melhoria das capacidades técnicas dos alunos
- Maior capacidade de experimentação e de adaptação
- Limitações técnicas com menos impacto nos resultados

Por todos os pontos acima, é fácil concluir que a aposta nas estratégias propostas, tem levado a uma melhoria substancial do trabalho dos alunos e do potencial dos seus projetos, servindo estes de portfólio demonstrativo da qualidade atual do curso e das capacidades dos alunos que está a preparar para o exigente mercado de trabalho.

4. Conclusões

Neste trabalho abordámos alguns desafios no ensino de Ciências da Computação a alunos de Videojogos. Propusemos uma resposta à questão sobre como pode a programação de Videojogos ser adequadamente lecionada num curso de 3 anos uniformemente interdisciplinar, como é o caso da Licenciatura em Videojogos da ULHT. A primeira parte desta resposta consistiu em definir os objetivos pretendidos, nomeadamente, promover o pensamento lógico e algorítmico dos alunos, bem como providenciar formas dos mesmos trabalharem fluidamente no *game engine* de eleição do curso. A partir daqui definimos uma abordagem *top-down*, na qual a seleção do *game engine* determina as metodologias educacionais a implementar, salientando três estratégias fundamentais: 1) desenvolvimento curricular holístico; 2) colaborações entre UCs de Ciências da Computação e de desenvolvimento de jogos; e, 3) adaptação de exemplos, exercícios e projetos à temática dos Videojogos. Os resultados preliminares mostram que a qualidade dos jogos produzidos nas UCs de

[12] Disponível em https://github.com/VideojogosLusofona.

[13] O site https://boardgamegeek.com/ é uma boa fonte de inspiração neste caso.

desenvolvimento de jogos, bem como a evolução técnica dos alunos, melhorou substancialmente, validando assim as estratégias implementadas.

Referências

Albahari, J., & Albahari, B. (2017). *C# 7.0 in a nutshell: The definitive reference* (First ed.). O'Reilly Media.

Astle-Adams, J. (2018, October). How to make games without programming. *Game-Dev.Net*. Retrieved from https://www.gamedev.net/articles/programming/general-and

-gameplay-programming/how-to-make-games-without-programming-r4987/ (Last accessed on 14/08/2019)

Astrachan, O. (2001). Oo overkill: When simple is better than not. In *Proceedings of the thirtysecond sigcse technical symposium on computer science education* (pp. 302–306). New York,

NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/364447.364608 doi: 10.1145/ 364447.364608

Bates, B. (2004). *Game design*. Premier Press.

Burns, B. (2008, January). Teaching the computer science of computer games. *Journal of Computing Sciences in Colleges*, *23*(3), 154–161. Retrieved from http://dl.acm.org/citation.cfm?id=

1295109.1295144

Cawley, C. (2015, August). How to make video games without any programming. *MakeUseOf* . Retrieved from https://www.makeuseof.com/tag/make-video-games-without-programming/ (Last accessed on 19/07/2019)

Claypool, K., & Claypool, M. (2005). Teaching software engineering through game design. In *Proceedings of the 10th annual sigcse conference on innovation e technology in computer science education* (pp. 123–127). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/ 10.1145/1067445.1067482 doi: 10.1145/1067445.1067482

Cleary, A., Vandenbergh, L., & Peterson, J. (2015). Reactive game engine programming for stem outreach. In *Proceedings of the 46th acm technical symposium on computer science education* (pp. 628–632). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/ 2676723.2677312 doi: 10.1145/2676723.2677312

Coleman, R., Krembs, M., Labouseur, A., & Weir, J. (2005). Game design & programming concentration within the computer science curriculum. In *Proceedings of the 36th sigcse technical symposium on computer science education* (pp. 545–550). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1047344.1047514 doi: 10.1145/1047344.1047514

Comber, O., Motschnig, R., Mayer, H., & Haselberger, D. (2019, April). Engaging students in computer science education through game development with unity. In *2019 ieee global engineering education conference (educon)* (pp. 199–205). doi: 10.1109/EDUCON.2019.8725135

Connolly, R. (2007, February). Teaching revulsion-free design patterns through game development. In *Proceedings of the 2007 microsoft academic days on game development in computer science education* (pp. 38–42).

Crytek. (2019). *CryEngine.* Retrieved from https://www.cryengine.com/

Dalal, N., Dalal, P., Kak, S., Antonenko, P., & S. S. (2009). Rapid digital game creation for broadening participation in computing e fostering crucial thinking skills. *International Journal of Social e Humanistic Computing*, *1*(2), 123–137. Retrieved from https://www.inderscienceonline .com/doi/abs/10.1504/IJSHC.2009.031002 doi: 10.1504/IJSHC.2009.031002

Dickson, P. E. (2015). Using unity to teach game development: When you've never written a game. In *Proceedings of the 2015 acm conference on innovation e technology in computer science education* (pp. 75–80). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/ 10.1145/2729094.2742591 doi: 10.1145/2729094.2742591

Dickson, P. E., Block, J. E., Echevarria, G. N., & Keenan, K. C. (2017). An experience-based comparison of unity e unreal for a stand-alone 3d game development course. In *Proceedings of the 2017 acm conference on innovation e technology in computer science education* (pp. 70–75).

New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3059009.3059013 doi: 10.1145/3059009.3059013

Dunn, F., & Parberry, I. (2011). *3D math primer for graphics e game development* (Second ed.).

CRC Press.

Epic Games. (2019). *Unreal engine.* Retrieved from https://www.unrealengine.com/

Fachada, N. (2018). Teaching database concepts to video game design e development students. *Revista Lusófona de Educação*, *40*(40), 151–165. Retrieved from http://revistas.ulusofona .pt/index.php/rleducacao/article/view/6437 doi: 10.24140/issn.1645-7250.rle40.10

Ficocelli, L., & Gregg, D. (2005). B.Sc. Computer game development... why not? In *Proceedings of digra 2005 conference: Changing views – worlds in play.*

Futschek, G. (2006). Algorithmic thinking: The key for understanding computer science. In R. T. Mittermeir (Ed.), *Informatics education – the bridge between using e understanding computers* (Vol. 4226, pp. 159–168). Springer Berlin Heidelberg. Retrieved from https://doi.org/ 10.1007/11915355_15 doi: 10.1007/11915355_15

Herbert, N., de Salas, K., Lewis, I., Dermoudy, J., & Ellis, L. (2014). ICT curriculum e course structure: The great balancing act. In *Proceedings of the sixteenth australasian computing education conference* (Vol. 148, pp. 21–30). Darlinghurst, Australia, Australia: Australian Computer

Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=2667490.2667493

Hewner, M., & Guzdial, M. (2010). What game developers look for in a new graduate: Interviews e surveys at one game company. In *Proceedings of the 41st acm technical symposium on computer science education* (pp. 275–279). New York, NY, USA: ACM. Retrieved from http:// doi.acm.org/10.1145/1734263.1734359 doi: 10.1145/1734263.1734359

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*,

9(3), 90–95. Retrieved from https://aip.scitation.org/doi/abs/10.1109/MCSE.2007.55 doi: 10.1109/MCSE.2007.55

Ip, B. (2012, April). Fitting the needs of an industry: An examination of games design, development, e art courses in the uk. *ACM Transactions on Computing Education*, *12*(2), 6:1–6:35. Retrieved from http://doi.acm.org/10.1145/2160547.2160549 doi: 10.1145/2160547.2160549

Johansson, T. (2018, October). *What is a job system?* Unity Blog. Retrieved from https:// blogs.unity3d.com/2018/10/22/what-is-a-job-system/ (Last accessed on 09/09/2019)

Jones, R. M. (2000). Design e implementation of computer games: A capstone course for undergraduate computer science education. In *Proceedings of the thirty-first sigcse technical symposium on computer science education* (pp. 260–264). New York, NY, USA: ACM. Retrieved from

http://doi.acm.org/10.1145/330908.331866 doi: 10.1145/330908.331866

Katz, D. (2013, January). *The unreasonable effectiveness of C.* Personal Blog. Retrieved from http:// damienkatz.net/2013/01/the_unreasonable_effectiveness_of_c.html (Last accessed on 01/08/2019)

Lengyel, E. (2012). *Mathematics for 3D game programming e computer graphics* (Third ed.). Cengage Learning.

Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. In *Proceedings of the 38th sigcse technical symposium on computer science education* (pp. 115– 118). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1227310 .1227352 doi: 10.1145/1227310.1227352

Linietsky, J., & Manzur, A. (2019). *Godot Engine.* Retrieved from https://godotengine.org/ Mateas, M., & Whitehead, J. (2007, February). Design issues for undergraduate game-oriented degrees. In *Proceedings of the 2007 microsoft academic days on game development in computer science education* (pp. 85–89).

McGugan, W. (2007). *Beginning game development with Python e Pygame: from novice to professional*. Apress.

Millington, I. (2019). *AI for games* (Third ed.). Boca Raton, FL, USA: CRC Press. Retrieved from https://doi.org/10.1201/9781351053303 doi: 10.1201/9781351053303

Morrison, B. B., & Preston, J. A. (2009). Engagement: Gaming throughout the curriculum. In *Proceedings of the 40th acm technical symposium on computer science education* (pp. 342–346).

New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1508865.1508990 doi: 10.1145/1508865.1508990

Mortoray, E. (2012, June). *What's to love about C?* Musing Mortoray. Retrieved from https:// mortoray.com/2012/06/11/whats-to-love-about-c/ (Last accessed on 12/08/2019)

Nystrom, R. (2014). *Game programming patterns*. Genever Benning. Retrieved from https:// gameprogrammingpatterns.com/

Parberry, I. (2011). Challenges e opportunities in the design of game programming classes for a traditional computer science curriculum. *Journal of Game Design e Development Education*, *1*, 4–17.

Parberry, I., Roden, T., & Kazemzadeh, M. B. (2005). Experience with an industry-driven capstone course on game programming: Extended abstract. In *Proceedings of the 36th sigcse technical symposium on computer science education* (pp. 91–95). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1047344.1047387 doi: 10.1145/1047344.1047387

Peng, C. (2015, December). Introductory game development course: A mix of programming e art. In *2015 international conference on computational science e computational intelligence (csci)* (pp. 271–276). doi: 10.1109/CSCI.2015.152

Reimer, J. (2005, August). Cross-platform game development e the next generation of consoles.

Ars Technica. Retrieved from https://arstechnica.com/features/2005/11/crossplatform/ (Last accessed on 14/08/2019)

Roden, T. E., & LeGrand, R. (2013). Growing a computer science program with a focus on game development. In *Proceeding of the 44th acm technical symposium on computer science education* (pp. 555–560).

Santos, A., Silva, J., & Junior, R. (2019). *Neokai.* Retrieved from https://videojogoslusofona

.itch.io/neokai (Last accessed on 02/10/2019)

Silicon Studio. (2019). *Xenko.* Retrieved from https://xenko.com/

Sociedade Portuguesa de Ciências dos Videojogos. (2019). *Ensino de videojogos.* Retrieved from http://www.spcvideojogos.org/ensino (Last accessed on 14/08/2019) Sousa, D., & Ferreira, J. (2018). *Hack of a plan.*

Spolsky, J. (2005, January). *Advice for computer science college students.* Joel on Software. Retrieved from https://www.joelonsoftware.com/2005/01/02/advice-for-computer-science -college-students/ (Last accessed on 15/09/2019)

Sturtevant, N. R., Hoover, H. J., Schaeffer, J., Gouglas, S., Bowling, M. H., Southey, F., ... Zabaneh, G. (2008). Multidisciplinary students e instructors: A second-year games course. In *Proceedings of the 39th sigcse technical symposium on computer science education* (pp. 383–387). New

York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/1352135.1352269 doi: 10.1145/1352135.1352269

Sweedyk, E., deLaet, M., Slattery, M. C., & Kuffner, J. (2005). Computer games e CS education: Why e how. In *Proceedings of the 36th sigcse technical symposium on computer science education* (pp. 256–257). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/ 10.1145/1047344.1047433 doi: 10.1145/1047344.1047433

Sweedyk, E., & Keller, R. M. (2005). Fun e games: A new software engineering course. In *Proceedings of the 10th annual sigcse conference on innovation e technology in computer science education* (pp. 138–142). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/

1067445.1067485 doi: 10.1145/1067445.1067485

Unigine Corp. (2019). *Unigine.* Retrieved from https://unigine.com/ Unity Technologies. (2018). *Unity* R . Retrieved from https://unity3d.com/

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22–30. Retrieved from https://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37 doi: 10.1109/MCSE.2011.37

Wick, M. R. (2005). Teaching design patterns in cs1: A closed laboratory sequence based on the game of life. In *Proceedings of the 36th sigcse technical symposium on computer science education* (pp. 487–491). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/

1047344.1047499 doi: 10.1145/1047344.1047499

Williams, S., & Spilman, T. (2019). *MonoGame.* Retrieved from http://www.monogame.net/

Wu, B., & Wang, A. I. (2012, January). A guideline for game development-based learning: A literature review. *International Journal of Computer Games Technology*, *2012*, 8:8–8:8. Retrieved from http://dx.doi.org/10.1155/2012/103710 doi: 10.1155/2012/103710

Yee, B., Sturman, D., & Feiner, S. (2007, February). Integrating video game development experience in an academic framework. In *Proceedings of the 2007 microsoft academic days on game development in computer science education* (pp. 28–32).

Zyda, M. (2006, June). Educating the next generation of game developers. *Computer*, *39*(6), 30–34.

Retrieved from https://ieeexplore.ieee.org/abstract/document/1642606 doi: 10.1109/ MC.2006.197

Zyda, M., Lacour, V., & Swain, C. (2008). Operating a computer science game degree program. In *Proceedings of the 3rd international conference on game development in computer science education* (pp. 71–75). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/

10.1145/1463673.1463688 doi: 10.1145/1463673.1463688